93065JOHNS809 D 2239

BYRON JOHNSON
356 LAGUNA TERR
SIMI VALLEY, CA 93065

Dr. Dobb's Journal
of Computer Calisthenics & Orthodontia

FREE SOFTWARE

COMPLETE SYSTEMS &
APPLICATIONS SOFTWARE
User documentation, internal specifications, annotated source code. In the two years of publication, *DDJ* has carried a large variety of interpreters, editors, debuggers, monitors, graphics games software, floating point routines and software design articles.

INDEPENDENT CONSUMER
EVALUATIONS

PRODUCT REVIEWS
& CONSUMER COMMENTS
Dr. Dobb's Journal publishes independent evaluations–good or bad–of products being marketed to hobbyists. It is a subscriber-supported journal. *Dr. Dobb's* carries no paid advertising; it is responsible *only* to its readers. It regularly publishes joyful praise and raging complaints about vendors' products and services.

REVIEWS

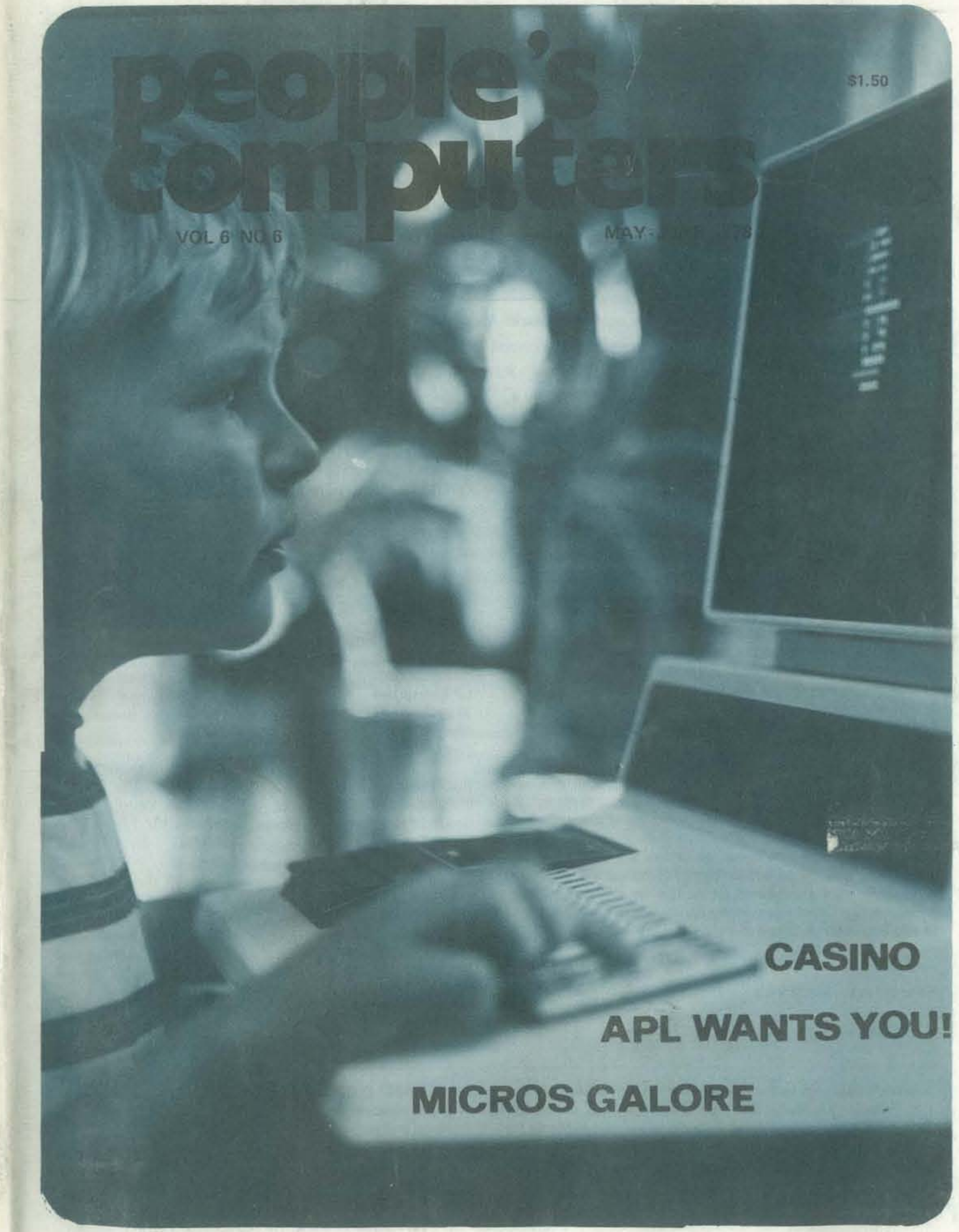
"A publication that is a must for everyone in the hobbyist world of computers. Don't miss it."
'Newsletter'
The Digital Group

"THE software source for microcomputers. Highly recommended."
'The Data Bus'
Philadelphia Area Computer Society

"It looks as if it's going to be THE forum of public domain hobbyist software development. Rating – ☆ ☆ ☆ ☆"
'TRACE'
Toronto Region Association of Computer Enthusiasts

"The best source for Tiny BASIC and other good things. Should be on your shelf."
'The Computer Hobbyist'
North Texas (Dallas) Newsletter

Dr. Dobb's Journal is published 10 times a year by People's Computer Company, a non-profit educational corporation. For a one-year subscription, send $12 to *Dr. Dobb's Journal*, Dept 81, 1263 El Camino Real, Box E, Menlo Park, CA 94025 or send in the postage-free order card at the center of this magazine.



people's computers

$1.50

VOL 6 NO 6

MAY-JUNE 1978

CASINO

APL WANTS YOU!

MICROS GALORE

SUBMITTING ITEMS FOR PUBLICATION

LABEL everything please, your name, address and the *date*; tapes should also include the program name, language and system.

TYPE text if at all possible, double-spaced, on 8½ x 11 inch white paper.

DRAWINGS should be as clear and neat as possible in black ink on white paper.

LISTINGS are hard to reproduce clearly, so please note:
- Use a new ribbon on plain white paper when making a listing; we prefer roll paper or fan-fold paper.
- Send copies of one or more RUNs of your program, to verify that it runs and to provide a sense of how things work – and to motivate more of us to read the code. RUNs should illustrate the main purpose and operation of your program as clearly as possible. Bells, whistles and special features should just be described in the documentation unless they're particularly relevant.
- Make sure your code is well documented – use a separate sheet of paper. Refer to portions of code by line number or label or address please, not by page number. When writing documentation, keep in mind that readers will include beginners and people who may be relatively inexperienced with the language you're using. Helpful documentation/annotation can make your code useful to more people. Documentation should discuss just which cases are covered and which aren't.
- If you send us a program to publish, we reserve the right to annotate it (don't worry, we won't publish it if we don't like it).
- Last but not least, please try to limit the width of your listings: 50-60 characters is ideal. Narrow widths mean less reduction, better readability and better use of space.

LETTERS are always welcome; we assume it's OK to publish them unless you ask us not to. Upon request we will withhold your name from a published letter, but we will not publish correspondence sent to us anonymously. We reserve the right to edit letters for purposes of clarity and brevity.

Cover photo courtesy of Visualscope.

SUBSCRIPTIONS

U. S. Subscriptions
- □ $8/yr. (6 issues)
- □ $15/2 yrs. (12 issues)
- □ Retaining subscription @ $25 ($17 tax deductible)
- □ Sustaining subscription @ $100+ ($92+ tax deductible)

Foreign Surface Mail
- □ add $4/yr. for Canada
- □ add $5/yr. elsewhere

Foreign AIRMAIL
- □ add $8/yr. for Canada
- □ add $11/yr. for Europe
- □ add $14/yr. elsewhere

Payment must be in U.S. dollars drawn on a U.S. bank.

These back issues are available at $1.50 each:
Vol 5, No 6
Vol 6, Nos 1, 2, 3, 4, 5

Foreign Distributors of *People's Computers*

Vincent Coen
LP Enterprises
313 Kingston Road
Ilford IG1 1PJ
Essex, UK

Rudi Hoess
Electronic Concepts PTY Ltd
Ground Floor Cambridge House
52-58 Clarence St
Sydney NSW 2000

ASCII Publishing
305 HI TORIO
5-6-7 Minami Aoyama
Minato-Ku, Tokyo 107
JAPAN

Home Computer Club
1070-57 Yamaguchi
Tokorozawa, Saitama,
JAPAN

Kougakusha Publ. Co., Ltd
Haneda Biru 403, 5-1
2-Chome, Yoyogi
Shibuya-Ku, Tokyo 151
JAPAN

Computer Age Co., Ltd
Kasumigaseki Building
3-2-5 Kasumigaseki
Chiyoda-Ku, Tokyo 100
JAPAN

People's Computers is published bimonthly by People's Computer Company, 1263 El Camino Real, Box E, Menlo Park, CA 94025.
People's Computer Company is a tax-exempt, independent, non-profit corporation, and donations are tax-deductible.
Second class postage paid at Menlo Park, California, and additional entry points.

Copyright © 1978 by People's Computer Company, Menlo Park, California

people's computers

VOL 6 NO 6
MAY-JUNE 1978

STAFF

EDITOR
Phyllis Cole
ASSISTANT EDITOR
Tom Williams
ART DIRECTOR
Meredith Ittner
PRODUCTION
Donna Lee Wood
ARTISTS
Maria Kent
Ann Miya
Judith Wasserman
TYPISTS
Barbara Rymsza
Sara Werry
BOOKSTORE
Dan Rosset
PROMOTION
Dwight McCabe
Andrea Nasher
CIRCULATION
Bill Bruneau
BULK SALES
Christine Botelho
DRAGON EMERITUS
Bob Albrecht

RETAINING SUBSCRIBERS

David R. Dick
Mark Elgin
John B. Fried
Scott Guthery, Computer Recreations
W. A. Kelley
John C. Lilly
Frank Otsuka
Bernice Pantell
Larry Press
Shelter Institute

SUSTAINING SUBSCRIBERS

Algorithmics Inc, Bruce Cichowlas
Don L. Behm
BYTE Publications, Carl Helmers, Virginia Peschke, Manfred Peschke
Paul, Lori and Tom Calhoun
Bill Godbout Electronics
Dick Heiser, The Computer Store

MICRO STUFF

ARTICLES

OTHER STUFF

REGULAR STUFF

EDITOR'S NOTES

This will be my last issue as editor. Future plans are not yet firm but they'll definitely include home computers. (No, I won't be working for Commodore. . .) Do not despair PET fans—SPOT will continue to appear in these pages.

People's Computers' new editor will be Bob Kahn, a long-time friend of People's Computer Company and, until taking on the editorship, a member of PCC's board of directors. When a junior in high school, back in 1962, Bob was first introduced to computers by none other than our Dragon Emeritus. He worked his way through college as a programmer and data analyst and spent a couple of years as a computer education consultant here in the San Francisco Bay Area. For the past 6 years, Bob has been the Director of the Computer Education Project at the University of California's Lawrence Hall of Science in Berkeley. At the same time, he has been working toward a Ph D in education at Berkeley. In addition to science museums and computers, Bob is very fond of kids, toasted almond ice cream, High Speed Ektachrome, and Renaissance dance music—not to mention The Dragons of Eden.

I'm looking forward to the fresh perspective Bob Kahn will bring People's Computers—it's sure to be enjoyable.

Phyllis Cole

LETTERS

I would like to say a few things about Pascal and Tiny Languages. All the good things said about Pascal are true. I have used a very powerful version of BASIC (BASIC-PLUS on the PDP-11) which could well be the best BASIC sold, but it is not as good as Pascal. However, Pascal is not usually interactive, and interactive languages have a lot of advantages over noninteractive ones.

Compared to other very powerful languages Pascal is small, but it is not tiny. I am helping to design and build a Pascal machine using the Z-80 (in contrast to putting Pascal on a Z-80), and find that according to our designs we can probably run a real-time disk operating system and Pascal compiler in 32K bytes. This is not tiny. Nevertheless we feel the power of Pascal is worth it since memory is not now expensive and will soon be even cheaper.

Pascal is very well designed. I wrote a compiler for Pascal before I ever wrote a program in Pascal and was surprised at how simple it was. I read books on parsing and compiler writing to prepare myself and then used a very primitive parsing algorithm, since the language is in a sense primitive. If a Tiny Language is similarly well designed then it can be powerful yet simple, but Pascal was never meant to be tiny, only small.

Pascal is as large as it is because it has many different data types and statements. A Tiny Language that has only one elementary data type, strings, as well as structured types, would be smaller. A simpler set of statements could still be used, such as assignment, a combination of the IF and CASE statements, and a looping statement. The simpler syntax would also make it easier to make the Tiny Language interactive yet still have free form.

Many people do not like to declare variables, but this is the essence of good programming style. BASIC's weakest feature is not its lack of structured control statements but its poor subroutine handling and lack of local variables. All variables should be declared, though defaults could be provided for beginning and lazy programmers.

An idea I have had for some time is to treat floating point numbers as fractions rather than as decimals. This would be very well suited to the string format and would also be easy for kids, as well as eliminating roundoff error.

Structured data types are even more important than structured control statements in my opinion, since powerful subroutine capabilities can reduce the need for powerful control statements. One of the problems with Pascal is that all array sizes are fixed at compile time. Dynamic arrays would be very nice. Records are also very useful in making programs easier to understand. Possibly all variables (simple, array, record and pointer), could be looked on as special cases of one variable type. Each variable could have some information indicating its size and pointers to its component strings. Since strings are of variable size there is really no difference from the implementer's point of view between a record and an array: both consist of a heading describing the variable and a string of pointers to the component strings. Referencing a field from a record would be exactly the same as referencing a component of an array.

Perhaps we are all guilty of cultural nearsightedness. All languages mentioned here are members of the Fortran/Algol family, which is what we have all been trained to program in. Rumor has it that some highly successful children's languages, such as Smalltalk, are entirely different. I wouldn't know, since no one has yet answered my requests for names of publications or ordering information. Since you seem to be in the know, how about reprinting relevant information? It would be a shame to develop a language that was outdated before it was even implemented.

My copies of Pascal News have just arrived. Anyone interested in Pascal should subscribe by sending $4.00 to Pascal User's Group, c/o Andy Mickel, University Computer Center: 227 EX, 208 SE Union Street, University of Minnesota, Minneapolis, MN 55455. One interesting bit of news was the announcement of Pascal implementations for microprocessors.

The University of California at San Diego has a Pascal system designed for Computer Aided Instruction written for the LSI-11, the 8080, the Z-80, and plan on having it run on the 6800 and the 6502, also. It requires at least 48K of memory, maybe more, and a certain number of floppy disks. The 8080 and Z-80 software uses the I/O drivers from CP/M, so if your system runs CP/M and has enough memory it should run UCSD Pascal. For more information write Pascal Group, Institute for Information Systems, UCSD Mailcode C-021, La Jolla, CA 92093.

A Pascal system for the 6800 was mentioned that requires 32K of memory and some high speed mass storage device such as a floppy disk or tape. The cost is around $100 and ordering information can be had from Computer Depot, 3515 West 70th Street, Minneapolis, MN 55435.

Both of the above systems come with complete source listings as well as other documentation. The UCSD system has a BASIC interpreter written in Pascal, CAI programs, text editors, and graphics capability.

The usual way to implement Pascal is to invent a hypothetical Pascal Machine which the compiler compiles. A small interpreter is written in the machine language of the computer to interpret the code of the Pascal Machine. To move the compiler to another machine it is only necessary to rewrite the interpreter, which is usually 4K to 8K. Thus most Pascal systems are very portable. The most popular series of compilers is the one started by Niklaus Wirth, called P1, P2, P3 and P4. The 6800 system is implemented using a variant of P2. UCSD seems to have invented theirs from scratch, although I could be wrong. I am using Per Brinck Hansen's Sequential Pascal/Concurrent Pascal pair of compilers because I am interested in real-time applications. It is possible for anyone to use one of the standard compilers to implement their own system in a few months. The best way to start is to subscribe to Pascal News.

Ralph Johnson
Galesburg, IL

In our Nov-Dec issue (Vol 6 No 3) we published references to Smalltalk (Alan Kay's article in the Sept 1977 Scientific American; *Kay and Goldberg's* Smalltalk Instruction Manual *from Xerox PARC, Palo Alto, CA). Our Jan-Feb issue (Vol 6 No 4) refers to Springer-Verlag books on Pascal (*Pascal User Manual and Report *by Jensen and Wirth; Ken Bowles'* Introduction to Computer Science*). Bowles' status report has been published in the March, 1978 issue of* Dr. Dobb's Journal *(Vol 3 No 3). Thanks for the other sources.*

I've met PASCAL recently and generally agree with David Mundie that it is a much better language than BASIC. However, there are a few problems that should not be ignored:

1. Character strings are not a basic variable type—the best that can be done is an array of individual characters. My mental processes work more easily with strings and substrings than with individual characters: I'd rather check for 'yes' than 'Y' and 'E' and 'S'.

2. Perhaps the problem is with the manual rather than the language, but I'm not sure exactly what can be read and written. I'm under the impression that only single characters can be read and written; apparently an internal number formatter was added as an afterthought (which does not inspire confidence).

3. Semicolons are required between every pair of statements—well *almost* every pair. I predict that users will find the misuse of semicolons to be the most persistant syntax problem. The only use I can see for semicolons is for separating several statements on the same line.

4. This may be nit-picking, but I don't consider the use of ':=' to be particularly clear. In addition it is unnecessarily clumsy to have to type two symbols for the most common operation. It seems to me that the use of a left-arrow for assignment would be a great improvement. If you can use an up-arrow in 'INPUT↑' (whatever that means) you can just as easily use a left arrow for assignment—it can't be that big a change.

5. Not only must statement labels be numbers instead of names (ugh), but each label must be declared in a LABEL statement (YUK!). Considering that PASCAL is nice enough to let me name my procedure, I fail to understand why I can't name my statements or why I must declare my labels before I use them. (For the fanatics who wish to eliminate GOTOs from the face of the earth, I refuse to make do without them simply because they can simplify an algorithm every once in a while.) In spite of my complaints, I still think PASCAL is a better language than BASIC.

For David Mundie: Please tell us the difference between an 'array' and a 'packed array'; and can you give a simpler example of a CASE statement? (Maybe I'm slow, but its use in your sample program was a bit shy of being crystalline).

For Bob Albrecht and Dennis Allison in particular: Before going much further with your tiny languages, I'd like to have some idea of the age group you are considering. (Would a six year old have any use for recursion or IF. . . THEN. . . ELSE?)

A similar question for graphics: are they to be controlled from the keyboard or from a user written program? An alternative to keyboard control would be special control knobs (e.g. for direction control) or a joy stick or something like that.

Occasionally I find myself deep in the middle of a bunch of REPEAT. . . UNTIL, WHILE, IF. . .THEN. . .ELSE, with a GOTO EXIT the only thing to be done at that point. Setting error flags and working my way out of all that logic to accomplish nothing more than that is unnecessarily complicated. Does that make me a poor programmer? That's my problem, not the language designer's.

Leigh Janes
East Lansing, MI

Dennis Allison has some comments for you, Leigh. 'On point 4, I agree. ASCII is, however, standard, and does not have enough graphics. I prefer = = for the assignment operator and = for comparisons. On point 5, the problem is historical, and comes from using labels in case statements. Further, lexical scoping and a desire to compute in one pass caused the need for declarations.'

I'm writing in direct reply to David Mundie's article in your January issue, and to comment indirectly on the spate of letters from the structured programming freaks. My feelings have gradually shifted from generally sympathetic to thoroughly annoyed, and I feel it's time to raise a few pertinent points.

If affordable general purpose computers are to become a commonplace, they must be purchased not by professional programmers nor by hobbyists, but by users with specialized non-trivial applications. The programs written by these users will not be widely distributed, nor will they be written for the love of intellectual exercise. They will be written to make one computer do something useful as quickly as possible. If this group of user-programmers fails to materialize, the 'computer revolution' is likely to produce only idiot-proof, preprogrammed appliance computers. I maintain that the user-programmer's first requirement is for fast convenient program development with a fully interactive editor-interpreter. Those people helping to shape the evolution of our programming languages ought to pay more attention to:

A. The difference between compiled and interpreted versions of any language.
B. The importance of the co-resident text editor in the design of any interpreter. (Could it be that many of the structured programming freaks are still 'editing' on keypunch machines?)
C. The degree to which any language is machine dependent, and particularly the influence of the ubiquitous teletype on the evolution of present interpreted languages.

Mr Mundie does not mention what system he used for 'BANBASIC', but he seems to be comparing a fairly powerful compiler with a severely restricted BASIC interpreter. Benchmarking the best of Hewlett-Packard's BASIC compilers against the first of the homebrew PASCAL interpreters would produce similar lopsided results. I believe it's time to forget about languages and talk about features.

Line numbers represent an economical means of implementing a text editor. Text editors will have to get much more powerful before we can afford to dispose of line numbers. Line numbers are also a convenient means of flagging errors. In a system without line numbers, the computer issuing an error message ought to display a sizable block of text and underline or highlight the offending section of code. Finally, line numbers label sections of code for an interpreter without requiring the interpreter to maintain a separate symbol table for entry points. Let me say here that an operating system with fully compatible interpreter and compiler, and a very good text editor, would remove most of my objections to the proposals of the structured programming people. I do not believe that such a system is feasible with our current crop of hardware.

The 'IF (condition) THEN (line number)' statement is the one that seems to annoy the S.P.F.'s the most. This has been almost universally replaced with the statement, 'IF (cond) (statement)/(statement) /(statement)'. There is no particular reason why this line cannot be extended to as many characters as desired, and listed back in the form,

```
IF (cond)
   THEN:
     statement
     statement
     statement
```

People should not waste time 'pretty-printing' while a computer which can do the same job more easily waits for them to finish! An 'ELSE' clause could be added in the same way, but should be optional. 'ELSE CONTINUE' is still the most common usage.

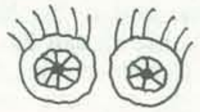
The unconditional 'GOTO' is obviously conditional in the construction above. It is also used in direct execution to test blocks of code without running an entire program. Furthermore, many existing editors do not permit renumbering or resequencing in any fashion—without the 'GOTO' many of us would spend more time retyping than programming. Best of all, it is cheap to implement. Use it or don't use it, but leave it in!

Multi-character variable names are a useful feature, but costly to implement in an interpreter. The business user will want them, but the engineering user might prefer faster lookup during run-time. Also, it is simply not true that meaningful variable names make a program easier to read—to be readily intelligible, an arithmetic expression must be physically compact. Try writing the general solution to a quadratic in linear form with 'meaningful' 8-character names for all constants and variables! In an arithmetic expression of any size, it is good practice to use very short variable names and describe those names in comment lines. This issue more than any other points up the difference between 'business' and 'scientific' languages. It will be difficult to please both groups of users with any single implementation of any language. I feel that BASIC clearly falls into the 'scientific' group of languages, and that we badly need a language doing for COBOL what BASIC did for FORTRAN. In the meantime, let's not 'reform' away BASIC's value as an easily implemented scientific language for very small machines.

Finally, Mr. Mundie says that BASIC encourages sloppy thinking. To this I say, deleted! Any language written for mass distribution must be extremely tolerant of sloppy thinking. The value of a computer is its ability to deal analytically with huge masses of rigorously structured data. The human mind is at its best while drawing loose analogies or metaphors, extracting patterns that cannot be demonstrated algorithmically. A computer should help people to order their thoughts—not require them to. The explicit declaration of all variables, the inability to branch freely in and out of procedural blocks, mandatory 'ELSE' and 'ENDIF' statements—all of these features will encourage 'logical thinking' and 'clean code'. They will also drive beginning users from the marketplace in droves.

Computers are to use, not to program!

David J. Beard
Newmanstown, PA



More letters, page 10

Heathkit's H-8 Computer System

BY TOM WILLIAMS

In 1975, pioneers trekked into the new world of personal computing which had been opened by the MITS Altair 8080. Most were hardy electronics hobbyists of the sort who hau previously found their outlet in amateur radio. Since then, the proliferation of personal computing has created two classes of users—the hobbyists and the consumer.

The hobbyist is interested in the *computer* and the consumer is interested in the *uses* for the computer. The hobbyist enjoys tinkering with the hardware and/ or software of his system in order to make it do all the 'neat' things he can think up for it, while the computer *consumer* is interested in buying a pre-assembled and tested machine with its operating software in ROM and as much preprogrammed applications software as possible.

HOMEWORK

Heathkit's H-8 system is an 8080-based computer aimed at the hobbyist, but with a difference; it is designed in such a way that new hobbyists are created out of some of the people who might not otherwise venture into hobbydom. However, Heath seems to be aware of the pitfalls of shattered expectations: the president's message in the 1977 Christmas catalog cautions that computers are not for everyone. In addition, notes packed in shipping cartons encourage the buyer to examine the manuals carefully before unpacking. Credit on a refund is offered if the customer decides he's bitten off more than he can chew or the system does not suit his needs.

There is no electronic kit of any sophistication that can be assembled by the complete novice. Most kits do not require a knowledge of electronics although some assume familiarity with components and procedures (such as the polarity of diodes and electrolytic capacitors). Heath has learned from experience not to take things for granted; I don't recommend that a totally inexperienced enthusiast attempt a computer kit and neither does Heathkit. Far better to cut your teeth on a stereo receiver or some such first.

A rudimentary knowledge of electricity is important both in building a kit as complex as a computer and in appreciating in some measure what is going on. Heath has built-in intermediate tests along the way which help you catch and isolate possible problems before you wind up confronting an inert mass of circuitry without knowing where to begin. Owning and knowing how to use a volt-ohmeter to measure voltage and resistance will help a great deal.

The H-8 computer system I constructed was a good sized task, requiring about 45 hours spread over a month and a half. I have previously constructed a number of Heath and other electronic kits ranging in complexity from multimeters to color televisions.

CONSTRUCTION

The system I built is advertised by Heath as 'System Two' and it consists of the H-8 computer, 16K of RAM, serial I/O and cassette board, audio cassette recorder and the H-9 video terminal. The assembly manual breaks down the imposing array of parts to manageable segments with a very clearly worded and illustrated set of instructions. The philosophy of the manuals seems to be that paper is cheap and mistakes are expensive. Manual changes and updates are included on separate sheets and you are instructed to collate these updates before starting construction.

Shortly before I started in on the H-9 terminal, I received a letter from Heathkit which contained not only the latest manual update, but also the piece of wire I would need to perform the required step!

For a person with some experience and confidence, constructing the various components is a straightforward task requiring little but care and patience. However, for those who either feel a bit uncertain or run into trouble, Heath maintains two islands of refuge. There are fifty Heath Electronic Centers located throughout the country. Each has factory-trained people who specialize in different areas of Heath products. Twice I had occasion to visit the store in Redwood City, CA and found the manager, Don Filmore, and his computer service technician, Dick DeCosta, both helpful and knowledgeable.

Of course, not everyone lives within easy distance of a Heathkit store and Heath has done what it can to help out here too. They maintain a telephone number in Benton Harbor, Michigan with technical advisers answering whatever desperate questions may come in from the hinterlands. I tried this number on four occasions—twice with real questions and twice with questions I had manufactured (naughty me). It was occasionally difficult to get through, but when I did, the woman who answered asked me how long I had been trying to reach them. Heath is responsive to the problem and is attempting to establish the proper size staff to handle telephone inquiries about their new product line.

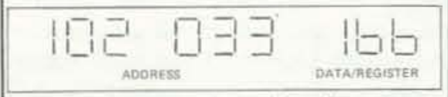
The technician I talked to had the manual, schematics and answers at his fingertips. The only nasty problem I had was a strange display on the H-9 terminal. The technician in Benton Harbor immediately recognized the area of the problem, and rattled off pairs of IC's for me to swap around to try to isolate it. When that didn't help I turned to the folks at the local store, who ultimately discovered a bad socket, a bad IC and (oops) a single solder bridge I had missed. Just as editors shouldn't proofread their own copy, so you should have someone *else* check over your soldering if you suspect a problem.

THE SYSTEM

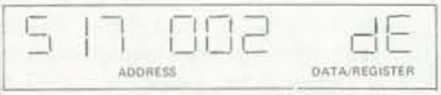
The completed H-8 computer looks different from some of the other 8080-based systems on the market—the front panel, for example. Whereas other systems have single LED's to represent address and data information, the H-8 has a nine-digit octal display. In the 'memory' mode, the first six digits display the address and the last three the contents of that address. Any address in memory can be selected from the front keypad and its contents examined and altered.

In the 'register' mode there is direct access to the internal CPU registers; you can examine them or alter their contents as you wish. The eight-bit registers are displayed in pairs and the sixteen-bit registers (stack pointer and program counter) are displayed individually.

A significant feature of the mainframe unit is the absence of a cooling fan. Each board has its voltage regulator IC's mounted to a heat sinking bracket on one end. This bracket is in turn secured to the chassis on the bottom and to a tie bracket on the top so that heat is dissipated both into the whole frame of the unit, and through vents in the top and bottom.



Display of contents of an address in memory



Display of contents of D and E CPU registers

There are two controversial features of the H-8. First, Heathkit has designed a completely new bus structure which it calls the Benton Harbor bus. It consists of 50 lines, all but seven of which have been permanently designated... and Heath *tells* you which ones it reserves the right to change. Heath has defended its choice of a new bus design on the grounds of better electronic characteristics and cost factors. It has been claimed that Heath, by having a unique bus structure, wanted to force the user to buy only Heath boards, but this argument doesn't hold up. It is to a manufacturer's advantage to have accessories generally available to the market. Even the biggest company can't produce everything at once and Heath's Director of Computer Products, Lou Frenzel, told me at the recent West Coast Computer Faire that they are hoping other small manufacturers *do* start producing boards compatible with the Benton Harbor bus. Also, a new bus structure cannot be marketed by just any company. The manufacturer has to be willing and able to wait for it to 'catch on' which can mean relatively slow initial sales. Thus, it looks as if Heath's decision to introduce the Benton Harbor bus was not made lightly and is a testimony to their faith in their design.

Secondly, there have been many discussions as to whether octal or hexidecimal notation is intrinsically better. Those who have chosen one or the other seem unshakable in their belief and this can get to be a pretty personal matter. I was weaned on hexadecimal and prefer it for a number of arbitrary reasons. However, my experience has been that it is easy enough to learn to operate within another number system. The only problem is that the majority of published software for microcomputers is in hexadecimal, so trying to work *between* the two systems can be tiresome. For this reason alone I wish Heath had chosen a hexadecimal display.

THE TERMINAL

The H-9 video terminal sharply and solidly displays twelve lines of eighty characters, upper case only . Most comparably priced terminals display more than twelve lines; evidently there was a tradeoff in order to obtain the 80 characters per line. As it turns out, twelve lines is sufficient for most purposes and the eighty column display can be quite handy.

The H-9 has three different display modes: the standard 12 x 80 (mentioned above), a short form which gives four 10 character columns, and a plot mode which puts a line across the middle of the screen and allows the display of simple graphs—that's graphs, *not* graphics. In addition, there is a full cursor control and a baud rate switch, which allows the user to choose between 300 baud and one other preselected speed from 110 to 9600 baud. One somewhat annoying aspect is that the 'return' key is the same size as all the others as well as being located next to the 'line feed' and 'scroll' keys. This, and the absence of lower case, are my only reservations on what otherwise appears to be an excellent terminal.

SOFTWARE

The H-8 is the first kit-form mainframe system to include a complete sytems software package in the price of the basic unit. In the past, computer kits would be sold with such things as IC sockets and all software at additional cost. With the H-8, all these are included, so there are no 'hidden costs.' A brief overview of the software follows:

PAM-8: This front panel ROM monitor allows control of the system through the front panel keypad. In addition to the display features described earlier, PAM-8 enables you to load or to dump from any desired port, to single step through a program using the single instruction key, and to reset the system logic. Heath's documentation also provides a complete listing of the monitor.

BUG-8: This console debugger allows entering and debugging machine language programs from a console terminal, displaying and altering memory and register contents, single instruction program execution, and tape load and dump routines.

TED-8: This text editor program allows writing source code for assembly language programming and configuring and editing text material for other purposes. TED-8 allows searching for a given string, editing it throughout the test or in specified lines.

HASL-8: This 8K assembler translates source code listings (provided by using TED-8) into absolute binary format which can be executed by the computer. HASL-8 can handle approximately 250 user-defined symbols.

Benton Harbor BASIC: This 8K BASIC comes with the system; the extended version of it is discussed briefly below.

For $20.00 Heath has available an extended version of Benton Harbor BASIC, written by Wintek Corporation. At first I was a bit skeptical about the label 'extended' since this BASIC resides in just a little over 9K of RAM, but after looking at the features, I find it quite satisfactory for its size. True, it is not as 'extended' as some 12K versions insofar as it does not have extensive editing and tracing features. It does, however, allow a number of operations not found in other BASICs of this size and *certainly* not at this price. The main improvements of the extended BASIC over that supplied with the computer are use of string functions, expanded math funtions, access to the real time clock, and a variety of CONTROL commands.

The FREE command tells not only how many bytes are free, but also how many are allocated for text, symbol table, FOR loops, GOSUB's, strings, and workspace. The STEP command will execute a program one line or a few lines at a time, and can be used as a primitive TRACE function. There is also a PORT command that will output the results of a PRINT statement to a port other than the console's port.

CONTROL commands are used to configure the size of the print zone, set up the front panel display to monitor a memory location or register during program execution, or to execute a specified GOSUB from the keyboard.

Both versions of Heath's BASIC represent numbers internally in floating point and are accurate to 6 digits. I found a noticeably weak point in exponentiation.

Raising a number (even an integer) to a power using the ↑ will not give a precise result because BASIC executes this command by multiplying the natural logarithm and then raising e to that power. Thus:(X ↑ Y) = EXP (Y * (LOG(X))

I talked to the people at Heath about this and they admitted it was a flaw. It is not a bug, but a tradeoff in the interest of memory space.

There is, however, one annoying aspect of all Heath software—'command completion.' Command completion means that as soon as the computer recognizes a command as unique, it automatically completes printing the rest of the command. This may be a convenience for those who type by the Columbus method (discover it and land on it) but in general it is a pain. If you type 'RU' the computer supplies the final 'N' but, more likely, a person will type the whole word and end up with 'RUN N' on the screen. Heath should be urged to supply software patches to make this feature optional.

Considering the features that are available in this extended BASIC for the size and price, I must say it's an exceptional value. In addition, the most recent version (which I haven't seen yet) also includes file capabilities—and its price tag is still $20.

DOCUMENTATION

The highest praise is justly reserved for last. I have already mentioned the assembly manuals. The operation manuals contain full schematics, timing diagrams, options for configuring I/O, instructions on the function of all keys, and detailed troubleshooting flowcharts. The electron-

H-8 PERIPHERALS

Heathkit will be introducing more accessories later in 1978. The ones we know of so far include:

	assembled	kit
H-17 Disk unit with 1 drive	$675 (June, 1978)	$575 (Fall, 1978)
extra drive	$295	
H-8-16 16K static Ram	$395 (Aug, 1978)	
H-8-7 breadboard for prototyping		$95 (Aug, 1978)

We have also heard of other peripherals soon to be available from various manufacturers, such as an S-100 adapter. We will report on these in the Accouncements section when we receive more information on them.

ically- interested user is able to study the theory of operation and circuit description sections to the extent of his interest.

The Software Reference Manual contains a detailed description of all the available software as well as a complete listing of the monitor and several BASIC utility routines. One of the most striking things about this documentation is this: it was prepared *before* the computer was actually marketed.

CONCLUSION

The combined quality of the hardware, the software and the documentation suggests that the H-8 system is an excellent learning device. The documentation is rich in detail and organized to help any reader find his own level, whether he is interested primarily in hardware or in software. The assembly procedures make kitbuilding accessible to those with limited to moderate experience. The organization of the front panel makes it possible to demonstrate clearly the machine's operation. The front panel should *not* be ignored by the beginner as something esoteric to be reserved for the advanced hacker, since it provides insight into the logic of the software and the structure of the machine.

With the H-8 system Heath has lived up to the reputation it has already established for quality in other kinds of electronic kits. The company's long experience in hi-fi, amateur radio and color television has given it the expertise required to produce a first-class piece of hardware. Engineering talent coupled with financial stability have given Heath the confidence to introduce design innovations. There is room for improvement in the software and Lou Frenzel is well aware of the need for more systems and applications software as well as the necessity of education the customers. At a recent convention in San Francisco he stated, 'The computer itself can be used as a teaching tool to help educate those people interested in computers. The idea is to provide computer aided instruction programs that individuals can use on their own computer to learn how to solve problems and program.' With this sort of awareness in the top management, I feel we can expect great things from Heath. □

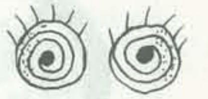
More LETTERS

continued from page 6.

The Computer Club at Coloma High School wants to act as a clearinghouse for microcomputer software for schools. This would give schools a chance to exchange programs and ideas, and to help other schools just getting started by sending them already working programs such as games, memory tests, grading programs and other such material. We are willing to act as a center to publish computer programs for schools willing to share in this idea and trade programs. Any interested hobbyists who have programs to share with schools would be welcomed.

In our center we have eight different microcomputer systems plus a 3M model 5500 test scorer. We can provide programs to share in 4 BASICs: the Poly extended version A00, Imsai CPM system BASIC-E version 1.33, Altair 8K BASIC version 4, and North Star BASIC. The storage systems we use are the Poly 88 Byte Base Cassette recording system, Imsai Dual Floppy Disk system with CPM, Tarbell Cassette recording system, North Star Mini Floppy Disk, and standard paper tape.

Terri Leamer
Coloma Computer Club
Coloma High School
Coloma, MI 49038

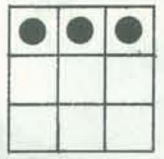


In the last issue of *People's Computers* (Vol 6 No 5, page 6) Jim Day made a comment about the game TEASER which I would like to dispute.

First, there are in fact exactly 102 possible positions in the game of TEASER (excluding rotations and reflections). At least, that is the answer I got and I've done the analysis three times.

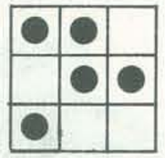
Second, there are in fact two errors in the diagram as published in *What to Do After You Hit Return* and in the September '74 *People's Computer Company*. If you examine the diagram you can see that the second board down (from the top) in the third column (counting from the left) and the fifth board down in the fifth column are identical. Likewise, the eighth and the eleventh boards up (from the bottom) in the fifth column (from the left) are identical. The first-mentioned board in each case should be replaced by Figures 1 and 2 respectively.

Figure 1



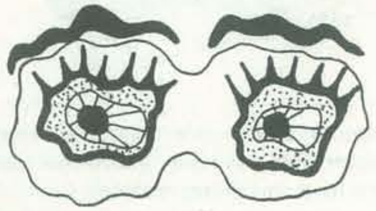
column 3 #2

Figure 2



column 5 #8

Eryk Vershen
Palo Alto, CA

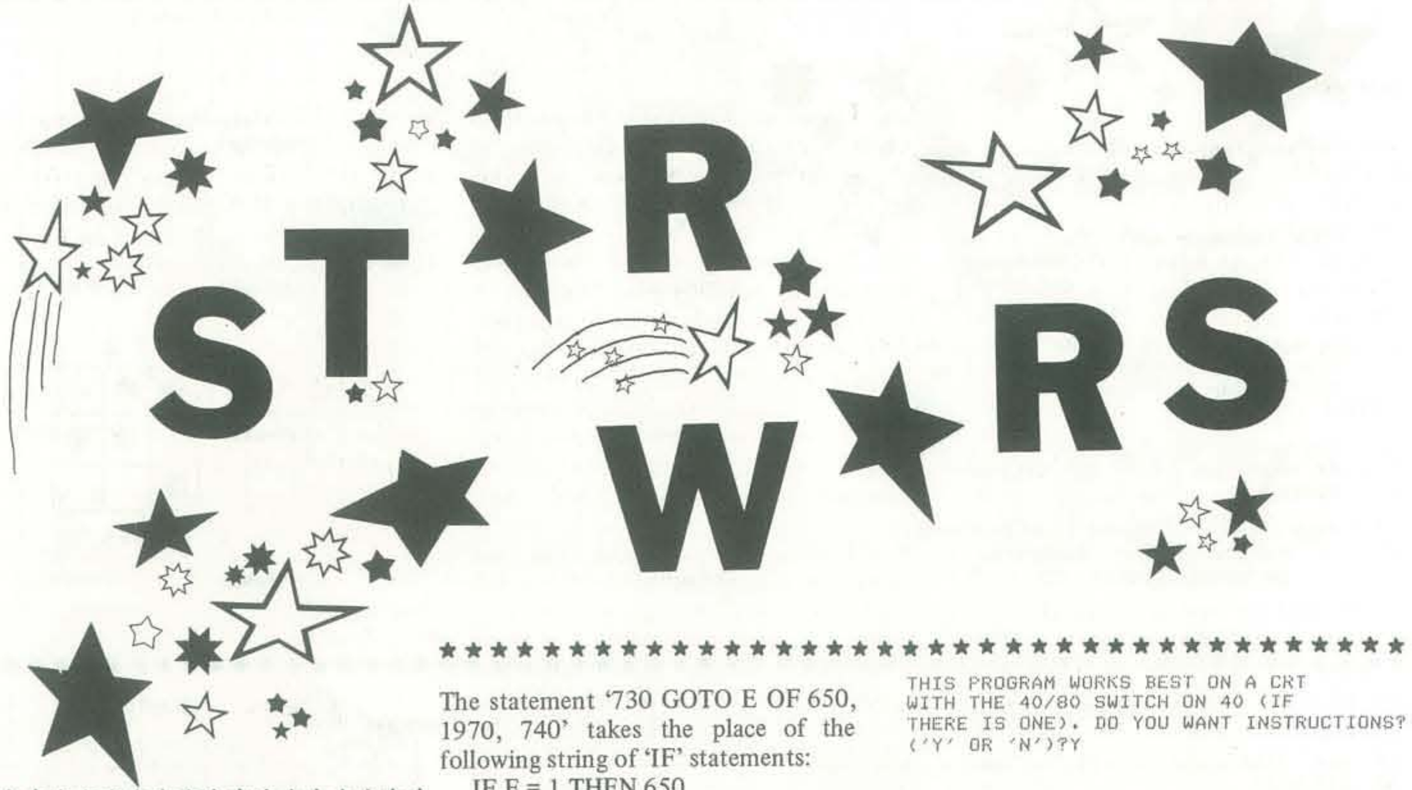


Do you remember the Digi-Comp I? It was an extremely simple mechanical computer made of plastic sliders, metal rods, and rubber bands, and it included a three (binary!) digit readout. The Digi-Comp was cycled with a manual clock and programmed by the placement of pins of various lengths along the sliders. Pins on one slider would activate or de-activate the rods, which pushed the pins on other sliders and changed the display. A later model, the Digi-Comp II, used balls rolling down a ramp and tripping flip-flops.

I believe the Digi-Comp was my introduction to the world of programming and logic. I never had the advanced model. Does anyone still have one? I can't help but think that today's kids of all ages might enjoy puzzling out how it works and trying to make it count from 0 to 7.

Kent Johnson
138 Hyde St. #19
San Francisco, CA 94102 □

STAR WARS



BY MARK PELCZARSKI

The movie STAR WARS suggests numerous game ideas for use with a computer. The real-time element of this attack on Death Star makes it particularly fun because, after all, the fate of the Galaxy is at stake. This game is by Mark Pelczarski, a teacher at Sycamore High School in Sycamore, Illinois. In addition to programming games, Mark has done a major project and thesis on CAI and Computer Managed Instruction at the University of Illinois.

My 'Star Wars' game is written for an HP2000 Access System (time-shared) computer. The statements which may have to be changed on other systems are the ENTER and computed GOTO statements. Lines 680 and 1870 are the critical ENTER statements (the others are just used as pauses). The statement 'ENTER L, N, M' will give the user L seconds to reply and put his/her reply in M (as INPUT M would). N is the amount of time allowed for a response—it is only used here to check if the reply was not answered quick enough, in which case N is negative (−256).

The statement '730 GOTO E OF 650, 1970, 740' takes the place of the following string of 'IF' statements:

IF E = 1 THEN 650
IF E = 2 THEN 1970
IF E = 3 THEN 740

Likewise, line '1370 GOTO M OF 1380, 1410, 1440, 1470, 1730, 1950, 1490' could be replaced with seven (or six) IF statements.

The program as it is in this version takes 2388 16-bit words on the HP2000. If you delete the rather lengthy instructions (lines 90-570) the length is cut down to 1193 words. Of course, some instructions should be kept in, but they can be much shorter.

This version of the game has 3 Tie Fighters. One is programmed to track you, the other two move around randomly and are a general nuisance. After the torpedo is chopped, you have to pull out or else you'll crash into the tower in front of you and never know what happened—whether you hit or missed.

This version is more challenging than my last one. I could beat it consistently before (however it was beaten only once at the 1-second interval). It took 3 or 4 serious runs at novice level (with a 5-second time limit) to produce the 'winning run' that starts on this page.

```
THIS PROGRAM WORKS BEST ON A CRT
WITH THE 40/80 SWITCH ON 40 (IF
THERE IS ONE). DO YOU WANT INSTRUCTIONS?
('Y' OR 'N')?Y


.    .  .   .  .  .  .  .   .  .   .  .
   .  .  .  .     .    .    .    .  .

.   .  .   .    .    .  .    .   .
  . . .    .     .     .     .    .

   .   .    .  .  .    .    .   . .
 .     . .     .    .      .    .

.   . .   . . . S  T  A  R .    . .   .
    . .  .  . .    .     .     .    .

 .   .    .  . .W  A  R  S  .   .   .  .
  .    .     .    .    .  .   .    .

  . ..   .   .     .  .  .   .   . .
   .   .   .   .   . .     .    .    .

  .    . .   . ..   .    .    .  .   . .
  ..  .  .    . .    .     .  .    .

.    . .   .   .   .    .   .    .. .
 . .     .   .   .  .    .    .    .


THE OBJECT, OF COURSE, IS TO DROP A
PROTON TORPEDO DOWN THE EXHAUST SHAFT
IN THE DEATH STAR.  YOU WILL START OUT
SPEEDING THROUGH THE CANYON, HOPING
THAT NO TIE FIGHTERS FIND YOU.   THEY
WILL.   YOU ARE TO TRY TO AVOID THEM BY
MANEUVERING BACK AND FORTH, UP AND DOWN,
UNTIL YOU SPOT THE SHAFT -- THEN FIRE!

YOU WILL BE SHOWN TWO VIEWS -- ONE
FORWARD (YOU'LL SEE AN 'X' WHERE YOU
ARE LOCATED) IN WHICH YOU'LL WATCH FOR
THE SHAFT, AND ONE BEHIND YOU, IN WHICH
YOU'LL LOOK FOR THE TIE FIGHTER (MARKED
AS AN 'H').  BOTH VIEWS ARE LOOKING
STRAIGHT THROUGH THE CANYON, WITH THE
WALLS AT YOUR SIDES AND THE CANYON FLOOR
BELOW.  THE SHAFT WILL FIRST APPEAR ON
THE CANYON FLOOR AS A '.' , BUT YOU
WON'T FIRE UNTIL YOU SEE IT AS A 'O'.
YOU MUST BE DIRECTLY ABOVE IT AND AS LOW
AS POSSIBLE.  DON'T FORGET TO PULL OUT
WHEN YOU SEE THE TOWER.
```

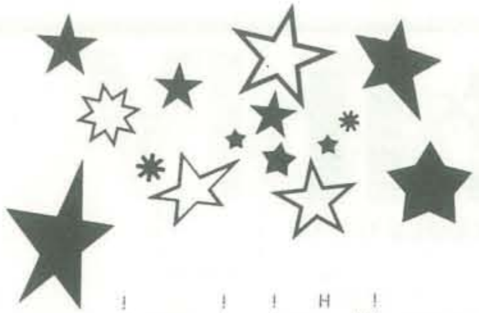
WINNING RUN

```
YOUR COMMANDS ARE:
        1) LEFT         4) RIGHT
        2) UP           3) DOWN
        5) FIRE         6) PULL OUT

BE SURE TO PRESS RETURN AS SOON AS YOU
GIVE YOUR COMMAND, OR IT WON'T BE
RECOGNIZED.  GOOD LUCK.


THE NUMBER YOU PICK FOR YOUR LEVEL WILL
BE YOUR TIME LIMIT (IN SECONDS) BETWEEN
MOVES.

YOUR LEVEL? (1-EXPERT, 2-VERY GOOD,
    3-GOOD, 4-FAIR, 5-NOVICE,
    20-SUPER NOVICE)  ?5
```

```
!   X    !  !        !
!        !  !        !
!        !  !        !
!--------!  !--------!
COMMAND?
3
!        !  !        !
!   X    !  !        !
!        !  !        !
!--------!  !--------!
COMMAND?
4
!        !  !        !
!    X   !  !        !
!        !  !        !
!--------!  !--------!
COMMAND?
4
!        !  !        !
!      X !  !        !
!        !  !        !
!--------!  !--------!
COMMAND?
4
```

```
!        !  !        !
!       X!  !        !
!        !  !        !
!--------!  !--------!
COMMAND?
3
!        !  !        !
!        !  !        !
!       X!  !        !
!--------!  !--------!
COMMAND?
1
!        !  !   H    !
!        !  !H       !
!     X  !  !      H !
!--------!  !--------!
COMMAND?
4
!        !  !    H   !
!        !  !H       !
!       X!  !H       !
!--------!  !--------!
COMMAND?
1
```

```
!        !  !      H !
!        !  !      H !
!      X !  !H       !
!--------!  !--------!
COMMAND?
1
!        !  !     H  !
!        !  !  H     !
!    X   !  !  H     !
!--------!  !--------!
COMMAND?
1
!        !  !   H    !
!        !  !     H  !
!   X    !  !      H !
!---.----!  !--------!
COMMAND?
1
!        !  !   H    !
!        !  !      H !
!   X    !  ! H      !
!---.----!  !--------!
COMMAND?
4
```



```
!        !  !   H    !
!        !  !       H!
!   X    !  !  H     !
!---O----!  !--------!
COMMAND?
5
!WWWWWWW!   !        !
!WWWWWWW!   !        !
!WWWWWWW!   ! H H H  !
!--------!  !--------!

COMMAND?
YOU'VE DONE IT -- A PERFECT SHOT!!!
CONGRATULATIONS!
WOULD YOU LIKE TO TRY AGAIN  ('Y' OR 'N') ?N
```

STAR WARS LISTING

```
10  REM  STAR WARS  -- 1977 -- MARK W. PELCZARSKI
20  DIM H[4]
30  PRINT "THIS PROGRAM WORKS BEST ON A CRT"
40  PRINT "WITH THE 40/80 SWITCH ON 40 (IF"
50  PRINT "THERE IS ONE). DO YOU WANT INSTRUCTIONS?"
60  PRINT "('Y' OR 'N')";
70  INPUT A$
80  IF A$="N" THEN 550
90  PRINT ""
100  PRINT ".     .  .    .    .   .   .    .  .      .    :
                .'  .     .  .         .      .    .      .
110  PRINT ".    .   .     .     .     .   .    .     .    .
                .'.  .      .       .        .       .
120  PRINT "     .   .       .     .    .      .      .  :  .
                .'         .   .        .  .       .
130  PRINT ".      .  .     .  .  .  S   T   A   R  .    .   .   ..
                .'.    .       : :      .    :      .
140  PRINT "  .     .        .   :  .W   A   R   S  .:      .     :  .
                .'     .       .       .     .     .    .
150  PRINT "    .  ..        .       .       .   .    .    .    . .
                .'      :    .      :      : '     .      :
160  PRINT "    .     . .     .   . .    :      .    .    :   . .
                .:    .   .       . .      .        .   .
170  PRINT ".      .  .      .    :      .      .     :      ..   .
                .'.        .     .     .     .      .     .      .
180  ENTER 10,M,N
190  PRINT ""          ← clear screen
200  PRINT "THE OBJECT, OF COURSE, IS TO DROP A"
210  PRINT "PROTON TORPEDO DOWN THE EXHAUST SHAFT"
220  PRINT "IN THE DEATH STAR.  YOU WILL START OUT"
230  PRINT "SPEEDING THROUGH THE CANYON, HOPING"
240  PRINT "THAT NO TIE FIGHTERS FIND YOU.   THEY"
250  PRINT "WILL.   YOU ARE TO TRY TO AVOID THEM BY"
260  PRINT "MANEUVERING BACK AND FORTH, UP AND DOWN,"
270  PRINT "UNTIL YOU SPOT THE SHAFT -- THEN FIRE!"
280  ENTER 10,N,M
290  PRINT "YOU WILL BE SHOWN TWO VIEWS -- ONE"
300  PRINT "FORWARD (YOU'LL SEE AN 'X' WHERE YOU"
310  PRINT "ARE LOCATED) IN WHICH YOU'LL WATCH FOR"
320  PRINT "THE SHAFT, AND ONE BEHIND YOU, IN WHICH"
330  PRINT "YOU'LL LOOK FOR THE TIE FIGHTER (MARKED"
340  PRINT "AS AN 'H').  BOTH VIEWS ARE LOOKING"
350  PRINT "STRAIGHT THROUGH THE CANYON, WITH THE"
360  PRINT "WALLS AT YOUR SIDES AND THE CANYON FLOOR"
370  PRINT "BELOW.  THE SHAFT WILL FIRST APPEAR ON"
380  PRINT "THE CANYON FLOOR AS A '.' , BUT YOU"
390  PRINT "WON'T FIRE UNTIL YOU SEE IT AS A 'O'."
400  PRINT "YOU MUST BE DIRECTLY ABOVE IT AND AS LOW"
410  PRINT "AS POSSIBLE.  DON'T FORGET TO PULL OUT"
420  PRINT "WHEN YOU SEE THE TOWER."
430  ENTER 10,N,M
440  PRINT "YOUR COMMANDS ARE:"
450  PRINT "        1) LEFT         4) RIGHT"
460  PRINT "        2) UP           3) DOWN"
470  PRINT "        5) FIRE         6) PULL OUT"
480  PRINT
490  PRINT "BE SURE TO PRESS RETURN AS SOON AS YOU"
500  PRINT "GIVE YOUR COMMAND, OR IT WON'T BE "
510  PRINT "RECOGNIZED.  GOOD LUCK."
520  PRINT "THE NUMBER YOU PICK FOR YOUR LEVEL WILL"
530  PRINT "BE YOUR TIME LIMIT (IN SECONDS) BETWEEN "
540  PRINT "MOVES."
550  PRINT "YOUR LEVEL? (1-EXPERT, 2-VERY GOOD,"
560  PRINT "    3-GOOD, 4-FAIR, 5-NOVICE,"
570  PRINT "    20-SUPER NOVICE)  ";
580  INPUT L
590  LET P=INT(RND(0)*13+8)
600  LET T=INT(RND(0)*4+2)
610  LET X1=E=C1=1
620  LET X2=4
630  LET H1=H2=C2=T1=S=0
640  LET C=-1
650  GOSUB 780
660  PRINT "COMMAND?";
670  PRINT
680  ENTER L,N,M
690  PRINT
700  IF N>0 THEN 720
710  LET M=7
720  GOSUB 1280
730  GOTO E OF 650,1970,740
740  PRINT "WOULD YOU LIKE ANOTHER RUN ('Y' OR 'N') ";
750  INPUT A$
760  IF A$="Y" THEN 590
770  GOTO 2000
780  REM - PRINT SCREEN -
790  LET C=C+1
800  PRINT ""      ← clear screen
810  FOR B=1 TO 3
820  GOSUB 890
830  NEXT B
840  IF C<P THEN 870
850  GOSUB 1210
860  RETURN
870  PRINT "!--------!   !--------!"
880  RETURN
890  REM - PRINT ROW -
900  PRINT "!";
910  IF X1 <> B THEN 970
920  LET A$="X"
930  LET A=X2
940  GOSUB 1120
950  PRINT "!   !";
960  GOTO 980
970  PRINT "        !   !";
980  LET A$="H"
990  IF H1 <> B THEN 1040
1000  LET A=H2
1010  GOSUB 1120
1020  PRINT "!"
1030  RETURN
1040  IF H1=0 THEN 1100
1050  LET H[C1+2]=B
1060  LET A=INT(RND(0)*7+1)
1070  LET H[C1]=A
1080  LET C1=C1+1
1090  GOTO 1010
1100  PRINT "        !"
1110  RETURN
1120  REM - LOCATE AND PRINT SHIP -
1130  FOR I=1 TO A-1
1140  PRINT " ";
1150  NEXT I
1160  PRINT A$;
1170  FOR I=1 TO 7-A
1180  PRINT " ";
1190  NEXT I
1200  RETURN
1210  REM - LOCATE SHAFT -
1220  LET C2=C-P+1
1230  GOTO C2 OF 1240,1240,1260
1240  PRINT "!---.---!   !--------!"
1250  RETURN
1260  PRINT "!---O---!   !--------!"
1270  RETURN
1280  REM - MOVE -
1290  IF C2<3 THEN 1370
1300  IF M <> 5 THEN 1350
1310  IF X1 <> 3 THEN 1800
1320  IF X2 <> 4 THEN 1800
1330  LET S=1
1340  GOTO 1800
1350  PRINT "YOU PASSED THE TARGET."
1360  LET T1=1
1370  GOTO M OF 1380,1410,1440,1470,1730,1950,1490
1380  LET X2=X2-1
1390  IF X2=0 THEN 1750
1400  GOTO 1490
1410  LET X1=X1-1
1420  IF X1=0 THEN 1780
1430  GOTO 1490
1440  LET X1=X1+1
1450  IF X1=4 THEN 1750
1460  GOTO 1490
1470  LET X2=X2+1
1480  IF X2=8 THEN 1750
1490  IF H2=0 THEN 1620
1500  FOR I=1 TO 2
1510  IF X2 <> H[I] THEN 1530
1520  IF X1=H[I+2] THEN 1570
1530  NEXT I
1540  LET C1=1
1550  IF X2 <> H2 THEN 1620
1560  IF X1 <> H1 THEN 1590
1570  PRINT "YOU'VE BEEN SHOT DOWN."
1580  GOTO 1760
1590  IF T1=1 THEN 1800
1600  LET H1=X1
1610  RETURN
1620  IF T1=1 THEN 1800
1630  IF H2 <> 0 THEN 1680
1640  IF C<T THEN 1670
1650  LET H1=1
1660  LET H2=4
1670  RETURN
1680  IF H2<X2 THEN 1710
1690  LET H2=H2-1
1700  RETURN
1710  LET H2=H2+1
1720  RETURN
1730  PRINT "YOU MISSED."
1740  GOTO 1490
1750  PRINT "YOU'RE GOING TO CRASH*"
1760  LET E=2
1770  RETURN
1780  PRINT "YOU'RE OUT OF RANGE, MOVE DOWN!"
1790  RETURN
1800  PRINT "!WWWWWWW!   !        !"
1810  PRINT "!WWWWWWW!   !        !"
1820  PRINT "!WWWWWWW!   ! H H H  !"
1830  PRINT "!--------!  !--------!"
1840  PRINT
1850  PRINT "COMMAND?";
1860  PRINT
1870  ENTER L,N,M
1880  IF M <> 6 THEN 1750
1890  IF S=0 THEN 1940
1900  PRINT "YOU'VE DONE IT -- A PERFECT SHOT!!!"
1910  PRINT "CONGRATULATIONS!"
1920  LET E=2
1930  RETURN
1940  PRINT "YOU MISSED YOUR TARGET."
1950  LET E=3
1960  RETURN
1970  PRINT "WOULD YOU LIKE TO TRY AGAIN  ('Y' OR 'N') ";
1980  INPUT A$
1990  IF A$="Y" THEN 550
2000  END
```

□

APPLE II

BY PHYLLIS COLE

Probably many of you are aware of the Apple—at both Computer Faires held so far their systems were showstoppers, though many winced at the price. The Apple II minimum configuration costs $1298 (without the color TV or cassette recorder).

Just what do you get for that? Well, the Apple II minimum configuration consists of a 6502-based system with a standard ASCII keyboard built into a lightweight (11 pound) housing that attaches to a home color (or black and white) TV. It comes with 8K of ROM (a 6K integer BASIC and a 2K monitor), 4K RAM, fast (1200 bps) interface for a standard cassette, and documentation. Despite Apple II's small size, there's room on board for two 2K socketed ROMs, up to 48K of RAM, and slots for 8 boards for peripherals.

Apple has announced 2 boards for intelligent peripherals. Their 'Intelligent Communications Interface' can be connected to any device which will accept a standard RS-232 interface, including the 103A-type modems. Features of the 'Intelligent Printer Interface' include the capability for printing up to 255 characters/line at 5000 characters/second, and an 8-bit parallel output port. No external power is required. Each board retails for $180. As of June, Apple expects to be shipping mini-floppy Shugart drives with an interface board that can support 2 floppy drives; no price had been announced as of mid-March. At the same time a revised Applesoft BASIC (10K or 12K, they're still squeezing the code down) will be available in ROM ($100) or on cassette tape ($15).

The Apple II uses Microsoft BASIC (which is rapidly becoming a de facto standard by virtue of its availability on so many systems—e.g. OSI Challenger, Commodore PET, Tandy TRS-80) and a screen editor. I didn't much like having to use 2 keys for cursor control (i.e. it takes 2 keystrokes to make the cursor move one space), but I suppose that's one of the prices you pay for having a standard keyboard. In text mode, the screen contains up to 24 lines of 40 characters each. No lower case letters can be displayed (a limitation left over from teletype days) which limits usefulness in educational environments and excludes word processing applications.

Two graphics modes are available. The 'normal' mode allows you to plot on a grid 40 cells wide and 48 cells high. In 'high resolution' mode a 280 wide and 192 high cell display is available. An optional 4 lines of text may be displayed at the bottom of the screen in either mode, thereby reducing grid size to 40 by 40 and 280 by 160, respectively. I saw high resolution mode (which requires a minimum of 16K of RAM) demonstrated in graphing a Bessel function. The plotting was impressively rapid, despite the fact that the BASIC program calculated in real time the location of each point.

Special BASIC commands allow you to select a color (15 are available in normal graphics mode, 4 in the high resolution mode); read the screen color at a given location; plot points, horizontal and vertical lines; read the game paddles.

A User software bank is being established by Apple; software for the system is also often available through computer magazines (e.g. *Kilobaud*, Jan & Feb, 1978). Software as now available for the system looks similar to that found on other comparable systems—disappointing. Apple has announced plans to remedy the program in part by field testing all software and documentation as one way of making sure only high quality materials are released. That's certainly a commendable first step, and one by which consumers will benefit. It's definitely time for manufacturers of consumer systems to turn more time, attention, and money to the problems of producing software and documentation.

Last but not least, a comment one one of Apple's recent ads. It joins a growing collection (alas) that should have died out long ago, picturing hubby in the kitchen with his computer, while proudly beaming housewife in the background washes dishes (or pares vegetables or performs other similar stereotypical tasks). Hey, how 'bout some reverse discrimination? If you insist on putting the computer in the kitchen, let's at least get the woman using it (no, *not* for the recipes—how about some fancy graphics program instead?) and let the husband do the dishes for a change! □

VideoBrain

BY PHYLLIS COLE

The current enthusiasm about home computers began in many garages and workshops across the country, as hundreds of electronics fans patiently wired together bits and pieces. At the time, it was the only way to obtain a reasonably priced home computer system; and only those with some hardware experience were likely to own a home system. Gradually the trend has been more and more towards home computers as consumer items. To date, the VideoBrain comes closest to presenting a computer as a consumer 'appliance', as evidenced by the fact that it's being carried in such stores as Macy's. The consumer does not program the machine, but merely inserts programs on ROM cartridges and then uses the minimal keyboard and up to 4 joysticks to provide input to pre-programmed games or the built-in function.

The VideoBrain sells for $499.95, which includes a keyboard console, 2 plug-in joysticks, an AC power adaptor, TV hookup cord and antenna switchbox, an owner's manual, and two introductory cartridge program packs, Music Teacher I and Wordwise I. The F-8 based system plugs into your home color (or black and white) TV. In addition to using ROM cartridge programs, the system may be used as a calculator. The user may also type, edit, store and retrieve a brief message and set an alarm.

VideoBrain is the name of the product by VideoBrain Computer Company, a subsidiary of Umtech, Inc, a Sunnyvale, California Company. Distribution of the system began early in 1978; there are plans to make additional programs available on a monthly basis. As of April, about a dozen such cartridges were available, ranging in price from $20 (Blackjact, Pinball, Wordwise II) to $30 (Gladiator, Math Tutor I, Dr Samuel's Checkers, Video Artist) to $50 (Financier) and $60 (Money Manager). Program size is generally reflected in the price; programs now available vary from 2K to 8K.

VideoBrain plans to provide programs in three main areas: entertainment, education, and home management. Of the programs I saw, by far the most sophisticated and interesting were those in the entertainment category. Gladiator, for example, is a 2-joystick, search-and-destroy type of game. But on the Gladiator cartridge there are 384 variations of the game with combinations of features such as bouncing objects, guided objects, fast objects, obstacle removal, speed control, and number of players. Last but not least, an overture and a finale played over your TV's speaker accompany the Gladiator package. The objects used in the game are created using a bit map display, and so the gladiators, lions, space ships, etc, are easily recognizable detailed pictures.

The VideoBrain is not well designed to support educational and home management programs. In particular, the limited amount of text that can be displayed, the lack of lower case characters, and the restricted keyboard are severe handicaps. Hopefully future versions of the system will be designed to better accomodate non-video games.

Some consumers may dislike having to rely on VideoBrain, at least for now, for pre-packaged programs. No software swaps are possible, since you're dealing with ROM cartridges. Even if you purchase the cassette recorder peripherals offered by the company, you're still faced with obtaining software in F-8 assembly language, and to date applications software for the F-8 is pretty much non-existent.

The concept underlying the VideoBrain is intriguing and attractive to the video game buff who likes the idea of having a system flexible enough to also be used for other purposes, but who has no interest in programming. If will be interesting to see what changes will occur in the product based on in-the-home experiences of consumers. □

In Music Teacher I, as you type in notes they're displayed on the screen's staff and played over the TV's speaker.

CASINO
A SMALL SYSTEM SIMULATOR

BY BØRGE CHRISTENSEN

In our Jan-Feb issue (Vol 6, No 4) Børge Christensen reported on COMAL, sometimes called 'Structured BASIC'. COMAL is a programming language developed by Christensen and his associates at DATO, a Teacher Training College in Tonder, Denmark. The language uses PASCAL-like control structures; Data General's Extended BASIC is a subset of COMAL.

This article is a tutorial on writing a simulator – the example used is a casino. The clear style and 'stepwise refinement' approach enable even non-programmers to understand the design of such a simulation.

Simulation is a problem-solving process, in which 'the actual system' or 'the problem system' is mapped onto a model, which most often takes the shape of a computer programme. Mathematical models answer questions such as 'What should I do under such and such circumstances?'. Unlike mathematical models, simulators answer to questions such as: 'If I act this way, how shall I expect the system to react?' ('If you push buttons A and C, will it pour out a Carlsberg for you or wash you away?').

Since simulation does not imply that the model is solvable in a mathematical sense, its applicability for practical purposes is much broader than pure mathematical techniques. Simulators have been designed to explain biological, psychological, sociological, economical, and other phenomena of the real world. Simulation has also been used to analyse systems in order to plan and explain them better. You might say that the simulator is built as a kind of 'exercise' to reveal the basic features of the actual system. As you design the simulator, you are forced to recognize what qualities of the different components are very important to the system as a whole, and how they interact with each other in it. In this article I'm going to describe a simulator of this last type.

I've chosen to simulate an imaginary casino. The reason for using such a fancy system is that I may thus keep it reasonably small and have some fun with the simulator afterwards. I was inspired to design it by reading about a similar system in Edwin R Sage's book, *Fun and Games with the Computer* (a very fine book; pity though that the author doesn't have a better language than BASIC as a vehicle to guide his bright thinking).

The description will fall in three parts: First, a general specification of the system is given; then as a second step, algorithms for the different parts of the simulator are developed by 'stepwise refinement', and finally the running program is presented.

In the following general specifications of the system, I have adopted a method developed by Lars Mathiassen of the University of Aarhus in Denmark. This method has been applied in setting up several simulators, one of which was used to analyse a large system employed by Danish hospitals for maintaining case records, including working conditions for the personnel attached to the system. My system is of course a humble one compared to that, but the description should nevertheless give you an impression of a method otherwise found extremely efficient.

CASINO *system:*

CROUPIER *component:*

data structure: is standing at the WHEEL facing the GAMBLER. May talk to the GAMBLER and spin the WHEEL.

action pattern: asks the GAMBLER to make his guess and then spins the WHEEL. The CROUPIER does so for each new game.

end CROUPIER.

WHEEL *component:*

data structure: the WHEEL is about 2 m diameter and is divided into 15 equal sectors. Five sectors are blue, four are green, three are yellow, two are black, and one is red.

action pattern: the WHEEL is spun by the CROUPIER and stops by itself after a few turns. When it has stopped, one of the fifteen sectors is opposite the pointer. The colour on that sector indicates the result of the game.

end WHEEL.

GAMBLER *component:*

data structure: the GAMBLER brings with him a certain amount of money, which he wants to stake hoping to win. The GAMBLER can see the WHEEL, see and hear the CROUPIER and the BANKER.

action pattern: the GAMBLER pays money to the BANKER, who opens an account for the GAMBLER. When the CROUPIER asks the GAMBLER to make his guess, he may pick out one of the colours on the WHEEL or he may quit the game. If he has selected a colour and thus indicated he wants to play on, he must make a bet with the BANKER. If the GAMBLER wants to leave, the BANKER will pay him the amount of money that remains in his account. If the GAMBLER doesn't behave properly he will be taken care of by the BOUNCER.

end GAMBLER.

BANKER *component:*

data structure: during the whole of the game the BANKER keeps up the GAMBLER's account. He can see the WHEEL and therefore knows the outcome of each game. He also knows the colour which the GAMBLER has picked out and the amount of money staked. He may activate the BELL and he may summon the BOUNCER.

action pattern: receives money from the GAMBLER and puts it down to his account when the GAMBLER arrives, and also if the GAMBLER has emptied his account during the games, but wants to continue. Before each game the BANKER records the GAMBLER's bet. If this bet exceeds the amount of money in the account, the BANKER will ask the GAMBLER to pay an amount into his account or make a less ambitious bet. If the GAMBLER refuses to do one or the other, the BANKER will ask him to leave. The BANKER will only accept a bet in whole dollar amounts.

When the WHEEL has stopped and the outcome of the game is available, the BANKER will subtract the bet from the GAMBLER's account if he has lost, or add the winnings to the account in case he has won. The winnings are calculated by the BANKER according to these rules: If blue wins, the BANKER pays 1 to 1 or 'even money',
if green wins, the BANKER pays 2 to 1,
if yellow wins, the BANKER pays 3 to 1,
if black wins, the BANKER pays 5 to 1, and
if red wins, the BANKER pays 12 to 1.

After updating, the BANKER informs the GAMBLER of the status of his account. If the GAMBLER wants to leave, the BANKER will normally thank him and invite him to come again soon. If the GAMBLER breaks a rule of the game, he is warned by the BANKER, and if the GAMBLER has received four such warnings, the BANKER will turn him over to the BOUNCER, who will take proper retaliatory measures. In this case, the BANKER will not thank the GAMBLER. An attempt from the side of the GAMBLER to overdraw his account will only be tolerated once; if he does so the BANKER gives him a special warning and in case of subsequent offence, the GAMBLER will be thrown out at once at the request of the BANKER.

end BANKER.

BOUNCER *component:*

data structure: very strong man, with good manners, though, and a persuasive bearing.

action pattern: on the BANKER's request he will ask the GAMBLER to leave the house without making further trouble.

BELL *component:*

data structure: electric bell. Is connected to a push button, which the BANKER alone may activate.

action pattern: rings each time the BANKER pushes the button.

end BELL

end CASINO *system.*

We shall now design algorithms for the different components, and for this purpose I'll apply a method known as 'stepwise refinement'. In this method you start by setting up a survey of the general structure of each component of the simulator. Many details are suppressed in the primary description and these details are then gradually introduced by refining the algorithms. It all ends up with a program of the simulator, which ought to answer the primary description of the system.

The component to look at first will be the GAMBLER. This is the *active* object of the system, it has a certain liberty whereas the other components are in fact restricted to *reacting* accordingly. Our first reflections shall therefore be dedicated to this component. We'll set up a catalogue of his activities:

he may get instructions for the game
he may put money into an account
he may guess a colour
he may make a bet
he may watch the outcome of the game
and have his account updated afterwards
he may leave the casino (one way or another).

Most of these activities are *conditional;* they are only carried out on certain assumptions. Let's look at them one by one: The GAMBLER should only get instructions for the game if he wants to. Maybe he's been to the CASINO before and knows all about the rules. Then he won't like to listen to detailed and–to him–boring explanations of the CASINO's favourite peculiarities. We therefore modify the first statement into:

```
if he wants then instruct him on the game
```

To be recognized as a GAMBLER, you must put money into an account. This is inevitable, so we leave the second statement as it is. During the negotiations with the BANKER concerning the opening of an account, the GAMBLER may become so unpopular with this important person that he is sent out. We shall have to modify the third statement into:

```
if he's not going to leave then he may guess a colour
```

During this part of the game the GAMBLER may–according to the rules–choose to leave the CASINO. The fourth statement is changed to:

```
if he's not going to leave then he may make a bet
```

While making a bet the GAMBLER may again come to blows with the BANKER over the rules; but if he makes an acceptable bet the two next activities are carried through. We thus may write:

```
if he's not going to leave then
    have the wheel spun and
    have the account updated
endif
```

Thus the prerequisite of the above mentioned activities is: the GAMBLER is not leaving. Correspondingly the precondition for leaving is that he *wants* to or that he *must*. At this level of description we shall be content with having established that he is leaving –for some reason or other:

```
if he's leaving then let him (one way or another)
```

Whether the GAMBLER is leaving or not is seen to be of crucial importance in all the cases we have set up. This in itself is not so peculiar–if there is no GAMBLER there is no game–but note that this precondition is in fact *the only one* we have to know about at this level. I therefore choose to represent that the GAMBLER is in a state of leaving–on his way out–by a flag (a Boolean variable): OUT, which is set (assigned a value of *true*) if the GAMBLER is leaving, and reset (has a value of *false*) if he's not. After having introduced this flag the GAMBLER's activities may with minor verbal modifications be stated like this:

```
if the gambler wants instruction then instruct him
allow him to put money into an account
if not out then have him guess a colour
if not out then have him make a bet
if not out then
    have the wheel spun
    have his account updated
endif
if out then take leave of him (one way or another).
```

Finally we note that some of the above mentioned activities are repeated as long as the game is running–(until the GAMBLER leaves) and so we program the various actions as *procedures* to get this final algorithm:

```
if the gambler wants instruction then exec instruction
exec account
repeat / / game is running / /
    if not out then exec guess
    if not out then exec bet
    if not out then
        exec wheel
        exec account
    endif
    if out then exec exit
until out
```

The running program is part of this article. In lines 280-390 you'll find the algorithm just designed in the shape of an actual running section of the program. As you can see I have preferred to let this section of the program serve as *main-program* (or monitor), since it represents the main component (the GAMBLER) and all his doings. Also notice that the flag OUT is reset from the start (line 170).

Before we go on looking at the various procedures, it may be convenient to put forward some general principles of this kind of program. When programming a simulator we shall have to *represent the states and quantities* appearing in the actual system in such a way that our computer may handle them. You also have to represent the possible *transitions between the states* of the system and the *conditions that control* these transitions. One might say that the *actions* and *decisions* must be mapped into the computer environment.

We also have to take into consideration that the simulator doesn't have the same *physical limitations* as the actual system. Thus it would not be possible in the real system for a gambler to bet on a colour not found on the wheel, but the operator at the terminal might very well enter an illegal response to one of the guess-requests of the simulator by some mistake or to provoke the program.

The first procedure to design would be *account (instruction* will simply print out a lot of text). A first approximation would be as simple as this:

```
proc account
    ask the gambler to invest
    put his money into his account
endproc account
```

But man is a frail creature, so there is a good chance that it won't be that easy. We shall have to foresee some of the nasty tricks the gambler might try with the BANKER, either because he misunderstood the rules of the CASINO or because he *is* tricky. First, the BANKER should not accept an investment of less than one whole dollar. Since the smallest legal bet is one whole dollar, it would not be suitable if the game allowed the GAMBLER to start with an inadequate amount–of say 65 cents. For another thing, there may be something wrong with the money the GAMBLER brings in: perhaps the currency is from some unknown country or has been made in the GAMBLER's own private works. All that and maybe more has to be looked after by the BANKER.

Since we are working with a simulator, our system does not accept real money as input (it would not be of much use to let our CRT try to swallow a $5 bill), but then again we have other problems. Imagine, say, some wise guy trying to type in a *negative* number, when asked to input the investment! (This is another example of the simulator not having the same physical limitations as the real system). What we must do, is of course to *interpret* the various possibilities in a proper way, and I've chosen to look at different *intervals* of the input to represent some possible situations in the real system. Using the variable *invest* to hold the input, I look at the following cases:

```
when    invest is negative
        tell the gambler his money is false; warn him
when    invest is zero
        tell the gambler to be serious; warn him
when    invest is positive, but less than one
        ask the gambler to come up with real money;
        warn him
when    invest is not whole, but greater than one accept it,
        but regard the fractional part as representing tips
        and only put the integer part into his account
otherwise
        accept the investment and put it into his account
```

Now, the BANKER will not have the GAMBLER sent out just because of a single mistake. He'll give him a chance to re-invest, until his investment is legal or he's finally had enough of him, i.e. he has given him at least four warnings. Thus we shall have to represent whether or not the investment is legal (the *state* of the investment) and the number of warnings given (a *quantity*). To represent the legitimacy of the investment, I've introduced another flag: OK, which is set, if the investment is acceptable, and reset, if it is not. To count the number of warnings, I use a numeric variable: WARNINGS. Although the number of warnings is a quantity, it is finally used to decide whether or not the GAMBLER is in a *state* of leaving. This state has already been represented by the flag OUT, and the transformation of the quantity WARNINGS into the state OUT may be done by using a statement like this:

```
if warnings > = 4 then out: = true
```

This transformation may, however, also be executed by a *Boolean function.* Since it may be that the reader has not seen such a device lately, I shall use it in this case, and define the Boolean function *t* by:

$$t(x) := (x >= 4)$$

If the argument x is greater than or equal to 4, the function will output a value of *true*, and for all other arguments, *t* will output a value of *false*. Finally the procedure looks like this:

```
proc account
    repeat / / get investment / /
        ok: = false
        ask the gambler to invest
        cases of invest:
        when    invest < 0
                tell the gambler his money is false
                warnings: = warnings + 1
        when    invest = 0
                tell the gambler to be serious
                warnings: = warnings + 1
        when    invest < 1
                ask the gambler to use real money
                warnings: = warnings+1
        when    invest <> int (invest)
                cents are tips, dollars are entered into account
                ok: = true
        otherwise
                ok: = true
        endcase
        out: = t (warnings)
    until ok or out
endproc account
```

The procedure is found as PROC ACCOUNT (lines 740-1030) in the program. It should be added that in the actual program the so-called 'otherwise case' is found between the CASE. . OF statement (line 790) and the first WHEN statement (line 820). The function FNT is defined in line 140 and used in line 1010.

I've set up the rest of the procedures using the same method as for PROC ACCOUNT, and I shall not go through them in detail, but rather restrict myself to a few remarks. I've tried to be very careful with variable-names and tests to make it possible for the reader, who has become familiar with the 'flag-representation' technique used in the MAINPROGRAM and in PROC ACCOUNT, to read the program.

Another flag, REALBAD, is introduced in PROC BADBET (subprocedure of BET) to represent the event that the GAMBLER has tried to overdraw his account. This flag is set in line 2010 at the same time as BET is being cancelled as too ambitious.

In line 380 you find the statement (mentioned above):

IF OUT THEN EXEC EXIT

We have already seen that OUT may be set for two reasons: the GAMBLER wants to leave or he is forced to leave because he has broken the rules of the game too often. In PROC EXIT we therefore have to examine why he got there. This is done in the statement in line 1080:

IF NOT FNT (WARNINGS) THEN

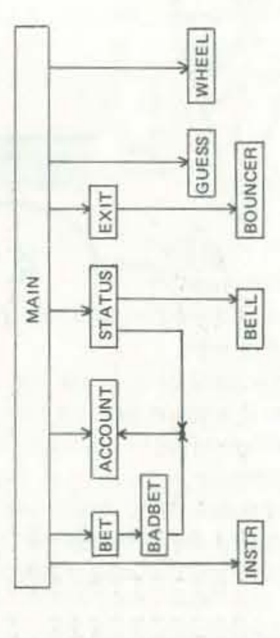
which is equivalent to 'If the GAMBLER did *not* come here because of too many *warnings* then. . .'. Thus the BOUNCER will only get hold of the GAMBLER if he has to leave because of four warnings or more (line 1140).

The flags OUT and REALBAD and the numeric variables WARNINGS, BET and ACCOUNT are also called the *attributes* of the GAMBLER, since between them they carry around the information necessary to offer this component a fair treatment. OUT and REALBAD are *global* flags, taking information from one procedure to another, whereas OK is a *local* flag, used for procedure-internal purposes only.

In this simulator one of the components—the GAMBLER—is stimulated from the world outside the computer. The person sitting at the terminal is an essential part of the GAMBLER, if not the gambler himself. Of course you can't control the outcome of the game, but anything else is up to you.

A different kind of simulator is the so-called *autonomous* one, which once started will run on controlled by its own internal structure only. This is a far more important class of simulator than the one presented in this article, but such simulators usually are more complicated too. In my next article I shall demonstrate how one can simulate some queue problems of a small supermarket by using the principles of autonomous components, controlled by random numbers only.

```
DO YOU WANT INSTRUCTIONS OF THE GAME? NO

HOW MUCH MONEY DO YOU WANT TO INVEST? 400


WHAT COLOUR DO YOU WANT TO BET ON?
BLUE/GREEN/YELLOW/BLACK/RED  BLUE

HOW MUCH DO YOU WANT TO BET?  40

*********************

      YELLOW

*********************

SORRY! YOU HAVE LOST YOUR BET, WHICH WAS $ 40
BETTER LUCK NEXT TIME!
YOU NOW HAVE $ 360 AT YOUR DISPOSAL.


WHAT COLOUR DO YOU WANT TO BET ON?
BLUE/GREEN/YELLOW/BLACK/RED  BLUE++++GREEN

HOW MUCH DO YOU WANT TO BET?  50

*********************

      BLUE

*********************

SORRY! YOU HAVE LOST YOUR BET, WHICH WAS $ 50
BETTER LUCK NEXT TIME!
YOU NOW HAVE $ 310 AT YOUR DISPOSAL.

                        .
                        .
                        .

SORRY! YOU HAVE LOST YOUR BET, WHICH WAS $ 50
BETTER LUCK NEXT TIME!
YOU NOW HAVE $ 10 AT YOUR DISPOSAL.
```



Structure of the Small Casino Program

```
WHAT COLOUR DO YOU WANT TO BET ON?
BLUE/GREEN/YELLOW/BLACK/RED  RED


HOW MUCH DO YOU WANT TO BET?  2

*********************

      GREEN

*********************


SORRY! YOU HAVE LOST YOUR BET, WHICH WAS $ 2
BETTER LUCK NEXT TIME!
YOU NOW HAVE $ 8 AT YOUR DISPOSAL.


WHAT COLOUR DO YOU WANT TO BET ON?
BLUE/GREEN/YELLOW/BLACK/RED  RED


HOW MUCH DO YOU WANT TO BET?  2

*********************

      RED

*********************

CONGRATULATIONS!
YOU HAVE WON $ 24 AND YOU NOW HAVE
$ 32 AT YOUR DISPOSAL.


WHAT COLOUR DO YOU WANT TO BET ON?
BLUE/GREEN/YELLOW/BLACK/RED  BLUE

HOW MUCH DO YOU WANT TO BET?  50
YOU HAVN'T GOT THAT MUCH MONEY!
DO YOU WANT TO INVEST SOME EXTRA MONEY? YES
```

```
HOW MUCH MONEY DO YOU WANT TO INVEST? 100

*********************

      BLUE

*********************

CONGRATULATIONS!
YOU HAVE WON $ 100 AND YOU NOW HAVE
$ 232 AT YOUR DISPOSAL.


WHAT COLOUR DO YOU WANT TO BET ON?
BLUE/GREEN/YELLOW/BLACK/RED  RED


HOW MUCH DO YOU WANT TO BET?  50

*********************

      BLUE

*********************

SORRY! YOU HAVE LOST YOUR BET, WHICH WAS $ 50
BETTER LUCK NEXT TIME!
YOU NOW HAVE $ 182 AT YOUR DISPOSAL.


WHAT COLOUR DO YOU WANT TO BET ON?
BLUE/GREEN/YELLOW/BLACK/RED  YELLOW


HOW MUCH DO YOU WANT TO BET?  50

*********************

      BLUE

*********************
```

```
SORRY! YOU HAVE LOST YOUR BET, WHICH WAS $ 50
BETTER LUCK NEXT TIME!
YOU NOW HAVE $ 132 AT YOUR DISPOSAL.


WHAT COLOUR DO YOU WANT TO BET ON?
BLUE/GREEN/YELLOW/BLACK/RED  BLUE


HOW MUCH DO YOU WANT TO BET?  200
YOU HAVN'T GOT THAT MUCH MONEY!
DO YOU WANT TO INVEST SOME EXTRA MONEY? NO

THEN YOU'LL HAVE TO BET LESS,
YOU ONLY HAVE $ 132 IN THE BANK
DON'T TRY TO OVERDRAW YOUR ACCOUNT.
THIS IS AN ULTIMATE WARNING!

HOW MUCH DO YOU WANT TO BET?  100

*********************

      BLUE

*********************

CONGRATULATIONS!
YOU HAVE WON $ 200 AND YOU NOW HAVE
$ 332 AT YOUR DISPOSAL.


WHAT COLOUR DO YOU WANT TO BET ON?
BLUE/GREEN/YELLOW/BLACK/RED  GREEN


HOW MUCH DO YOU WANT TO BET?  400

YOUR PRESENCE IN THE CASINO IS NOT WANTED
PLEASE LEAVE THIS HOUSE WITHOUT ANY TROUBLE.

THE CONTENTS OF YOUR ACCOUNT, TOTAL $ 332
IS RETURNED FROM THE DESK AT THE ENTRANCE.
```

```
0010 REM (*THIS IS THE SIMULATOR: SMALL CASINO*)
0020 REM (*WRITTEN FOR 'PEOPLE'S COMPUTERS*)
0030 REM (*BY BØRGE R. CHRISTENSEN*)
0040 REM (*AT 'DATO', TONDER, DENMARK*)
0050 REM (*DATE OF THIS VERSION: FEB. 19. 1978*)
0060 REM
0070 REM //---------------//
0080 REM
0090 REM (*INIT*)
0100 REM
0110 RANDOMIZE
0120 REM ** BOOLEAN CONSTANTS: TRUE AND FALSE ARE DEFINED **
0130 LET TRUE=1; FALSE=0
0140 DEF FNT(X)=(X>=4)
0150 REM //---------------//
0160 REM ** ATTRIBUTES OF GAMBLER ARE INITIALIZED **
0170 LET OUT=FALSE; REALBAD=FALSE
0180 LET WARNINGS=0; BET=0; ACCOUNT=0
0190 REM //---------------//
0200 REM ** UTILITY STRINGS ARE DECLARED **
0210 DIM ANSW$(5),COLOUR$(6),OUTCOME$(6)
0220 REM //---------------//
0230 REM (*ENDINIT*)
0240 REM //---------------//
0250 REM
0260 REM (*MAINPROGRAM*)
0270 REM
0280 INPUT "DO YOU WANT INSTRUCTIONS OF THE GAME? YES/NO ",ANSW$
0290 IF ANSW$(1)="Y" THEN EXEC INSTR
0300 EXEC ACCOUNT
0310 REPEAT
0320   IF NOT OUT THEN EXEC GUESS
0330   IF NOT OUT THEN EXEC BET
0340   IF NOT OUT THEN
0350     EXEC WHEEL
0360     EXEC STATUS
0370   ENDIF
0380   IF OUT THEN EXEC EXIT
0390 UNTIL OUT
0400 END OF MAIN
0410 REM
0420 REM //---------------//
0430 REM
0440 REM (*PROCEDURES*)
0450 REM
0460 REM //---------------//
0470 REM
0480 PROC GUESS
0490   PRINT
0500   LET OK=FALSE
0510   REPEAT
0520     PRINT
0530     PRINT "WHAT COLOUR DO YOU WANT TO BET ON?"
0540     INPUT "BLUE/GREEN/YELLOW/BLACK/RED  ",COLOUR$
0550     CASE COLOUR$ OF
0560       PRINT
0570       PRINT "OPERATING ERROR! IMPOSSIBLE SITUATION!"
0580       INPUT "DO YOU WANT INSTRUCTIONS? YES/NO ",ANSW$
0590       IF ANSW$(1)="Y" THEN EXEC INSTR
0600     WHEN "NONE"
```

```
0610        LET OUT=TRUE
0620      WHEN "BLUE","GREEN","YELLOW","BLACK","RED"
0630        LET OK=TRUE
0640      ENDCASE
0650    UNTIL OUT OR OK
0660 ENDPROC GUESS
0670 REM
0680 REM //---------------//
0690 REM
0700 REM (*THE FOLLOWING PROCEDURES ARE BANKERS TASKS*)
0710 REM
0720 REM //---------------//
0730 REM
0740 PROC ACCOUNT
0750   REPEAT
0760     LET OK=FALSE
0770     PRINT
0780     INPUT "HOW MUCH MONEY DO YOU WANT TO INVEST? ",INVEST
0790     CASE TRUE OF
0800       LET ACCOUNT=ACCOUNT+INVEST
0810       LET OK=TRUE
0820     WHEN INVEST<0
0830       PRINT
0840       PRINT "KEEP YOUR FALSE MONEY - Y!"
0850       LET WARNINGS=WARNINGS+1
0860     WHEN INVEST=0
0870       PRINT
0880       PRINT "I HAD THE IMPRESSION, YOU MEANT THIS SERIOUSLY!"
0890       LET WARNINGS=WARNINGS+1
0900     WHEN INVEST<1
0910       PRINT
0920       PRINT "NO SIR!! NOT THAT CENT STUFF. REAL MONEY PLEASE!"
0930       LET WARNINGS=WARNINGS+1
0940     WHEN INVEST<>INT(INVEST)
0950       PRINT
0960       PRINT "TIPS! YOU  A R E  GENEROUS, SIR!"
0970       LET INVEST=INT(INVEST)
0980       LET ACCOUNT=ACCOUNT+INVEST
0990       LET OK=TRUE
1000     ENDCASE
1010     LET OUT=FNT(WARNINGS)
1020   UNTIL OK OR OUT
1030 ENDPROC ACCOUNT
1040 REM
1050 REM //---------------//
1060 REM
1070 PROC EXIT
1080   IF NOT FNT(WARNINGS) THEN
1090     PRINT
1100     PRINT "THANKS FOR THE GAME"
1110     IF WARNINGS<2 THEN PRINT "IT´S BEEN A PLEASURE."
1120     PRINT "COME BACK AGAIN SOME DAY"
1130   ELSE
1140     EXEC BOUNCER
1150   ENDIF
1160   IF ACCOUNT<>0 THEN
1170     PRINT
1180     PRINT "THE CONTENTS OF YOUR ACCOUNT, TOTAL $";ACCOUNT
1190     PRINT "IS RETURNED FROM THE DESK AT THE ENTRANCE."
1200   ENDIF
1210 ENDPROC EXIT
1220 REM
1230 REM //---------------//
1240 REM
1250 PROC STATUS
1260   IF COLOUR$=OUTCOME$ THEN
1270     LET ACCOUNT=ACCOUNT+BET*FACTOR
1280     EXEC BELL
1290     PRINT
1300     PRINT "CONGRATULATIONS!"
1310     PRINT "YOU HAVE WON $";BET*FACTOR;"AND YOU NOW HAVE"
1320     PRINT "$";ACCOUNT;"AT YOUR DISPOSAL."
1330   ELSE (*YOU HAVE LOST*)
1340     LET ACCOUNT=ACCOUNT-BET
1350     PRINT
1360     PRINT "SORRY! YOU HAVE LOST YOUR BET, WHICH WAS $";BET
1370     PRINT "BETTER LUCK NEXT TIME!"
1380     PRINT "YOU NOW HAVE $";ACCOUNT;"AT YOUR DISPOSAL."
1390     IF ACCOUNT=0 THEN
1400       INPUT "DO YOU WANT TO INVEST MORE MONEY? YES/NO ",ANSW$
1410       IF ANSW$(1)="Y" THEN
1420         EXEC ACCOUNT
1430       ELSE (*NO, I´VE HAD ENOUGH*)
1440         LET OUT=TRUE
1450       ENDIF
1460     ENDIF
1470   ENDIF
1480 ENDPROC STATUS
1490 REM
1500 REM //---------------//
1510 REM
1520 PROC BET
1530   PRINT
1540   PRINT
1550   REPEAT
1560     LET OK=FALSE
1570     PRINT
1580     INPUT "HOW MUCH DO YOU WANT TO BET?  ",BET
1590     CASE TRUE OF
1600       REM (*BET IS OK*)
1610       LET OK=TRUE
1620     WHEN BET>ACCOUNT
1630       EXEC BADBET
1640     WHEN BET<>INT(BET)
1650       PRINT
1660       PRINT "THIS IS NO GAME OF CENTS. MEANY!"
1670       LET WARNINGS=WARNINGS+1
1680     WHEN BET<=0
1690       PRINT
1700       PRINT "DON´T WASTE OUR TIME!"
1710       LET WARNINGS=WARNINGS+1
1720     ENDCASE
1730     LET OUT=FNT(WARNINGS)
1740   UNTIL OK OR OUT
1750 ENDPROC BET
1760 REM //---------------//
1770 PROC BADBET
1780   IF REALBAD THEN
1790     LET WARNINGS=4
1800   ELSE
1810     PRINT "YOU HAVN´T GOT THAT MUCH MONEY!"
1820     REPEAT
1830       LET OK=FALSE
1840       INPUT "DO YOU WANT TO INVEST SOME EXTRA MONEY? YES/NO ",ANSW$
1850       IF ANSW$(1)="Y" THEN
1860         EXEC ACCOUNT
1870         IF BET>ACCOUNT THEN
1880           PRINT
1890           PRINT "YOUR BET STILL EXCEEDS YOUR ACCOUNT!"
1900           PRINT "THE CASINO DOES CERTAINLY NOT APPROVE OF SUCH MANNERS!"
1910           PRINT
1920           LET WARNINGS=WARNINGS+2
1930         ENDIF
1940       ELSE (*NO, YOU BET I WON´T*)
1950           PRINT
1960           PRINT "THEN YOU´LL HAVE TO BET LESS,"
1970           PRINT "YOU ONLY HAVE $";ACCOUNT;"IN THE BANK"
1980           PRINT "DON´T TRY TO OVERDRAW YOUR ACCOUNT."
1990           PRINT "THIS IS AN ULTIMATE WARNING! "
2000           LET WARNINGS=WARNINGS+1
2010           LET REALBAD=TRUE; BET=0
2020         ENDIF
2030         LET OUT=FNT(WARNINGS)
2040     UNTIL BET<=ACCOUNT OR OUT
2050   ENDIF
2060 ENDPROC BADBET
2070 REM
2080 REM //---------------//
2090 REM
2100 REM ** END OF BANKER COMPONENT **
2110 REM
2120 REM //---------------//
2130 REM
2140 PROC WHEEL
2150   LET N=RND(0)
2160   CASE TRUE OF
2170     LET OUTCOME$="RED"; FACTOR=12
2180   WHEN N<1/3
2190     LET OUTCOME$="BLUE"; FACTOR=2
2200   WHEN N<3/5
2210     LET OUTCOME$="GREEN"; FACTOR=3
2220   WHEN N<4/5
2230     LET OUTCOME$="YELLOW"; FACTOR=4
2240   WHEN N<14/15
2250     LET OUTCOME$="BLACK"; FACTOR=6
2260   ENDCASE
2270   PRINT
2280   PRINT "*********************"
2290   PRINT
2300   PRINT "      ";OUTCOME$
2310   PRINT
2320   PRINT "*********************"
2330 ENDPROC WHEEL
2340 REM
2350 REM //---------------//
2360 REM
2370 PROC BELL
2380   FOR I=1 TO 2*FACTOR
2390     PRINT "<7>";
2400   NEXT I
2410 ENDPROC BELL
2420 REM
2430 REM //---------------//
2440 REM
2450 PROC BOUNCER
2460   PRINT
2470   PRINT "YOUR PRESENCE IN THE CASINO IS NOT WANTED"
2480   PRINT "PLEASE LEAVE THIS HOUSE WITHOUT ANY TROUBLE."
2490   LET OUT=TRUE
2500 ENDPROC BOUNCER
2510 REM
2520 REM //---------------//
2530 REM
2540 PROC INSTR
2550   PRINT
2560   PRINT "THIS IS THE GAME:  S M A L L   C A S I N O"
2570   PRINT
2580   PRINT "IN THIS GAME IS A   C R O U P I E R ,  A  B A N K E R ,  "
2590   PRINT "A  B O U N C E R  AND  A  W H E E L."
2600   PRINT
2610   PRINT "THE WHEEL IS SECTORED IN 15 PIE-SHAPED COLOURED PARTS. "
2620   PRINT "5 SECTORS ARE BLUE, 4 SECTORS ARE GREEN, 3 SECTORS ARE YELLOW"
2630   PRINT "2 SECTORS ARE BLACK, AND 1 SECTOR IS RED."
2640   PRINT "THE WHEEL IS SPUN BY THE CROUPIER, AND WHEN IT STOPS,"
2650   PRINT "ONE OF THE COLOURS IS POINTED TO AS THE OUTCOME OF THE GAME."
2660   PRINT
2670   INPUT "WHEN YOU HAVE READ THIS, STRIKE ´RETURN´  ",ANSW$
2680   PRINT
2690   PRINT "IF YOU WANT TO PLAY IN THIS GAME, THE BANKER WILL ASK YOU TO INVEST"
2700   PRINT "SOME MONEY. SMALLEST INVESTMENT IS 1 DOLLAR. IF YOU GIVE HIM DOLLARS"
2710   PRINT "AND CENTS, HE´LL THINK, THE CENTS ARE TIPS, AND ONLY THE AMOUNT"
2720   PRINT "OF DOLLARS WILL BE ADDED TO YOUR ACCOUNT."
2730   PRINT "IF SOMEHOW DURING THE GAME YOU EMPTY YOUR ACCOUNT, THE BANKER"
2740   PRINT "WILL ASK YOU TO INVEST SOME MORE MONEY. IF YOU DON´T WANT TO,"
2750   PRINT "YOU MUST LEAVE THE CASINO."
2760   PRINT
2770   PRINT "AS SOON AS YOU HAVE INVESTED, YOU ARE RECOGNIZED AS A GAMBLER,"
2780   PRINT "AND YOU WILL BE ASKED BY THE CROUPIER TO MAKE A GUESS."
2790   PRINT "THERE ARE FIVE POSSIBLE GUESSES: BLUE, GREEN, YELLOW,"
2800   PRINT "BLACK, AND RED. ONLY THESE FIVE COLOURS ARE ACCEPTED."
2810   PRINT "IF YOU WANT TO LEAVE, JUST ANSWER ´NONE´, WHEN YOU ARE"
2820   PRINT "ASKED TO PICK OUT A COLOUR"
2830   PRINT "ANY OTHER ANSWER THAN ONE OF THE FIVE COLOURS OR ´NONE´"
2840   PRINT "IS CONSIDERED ILLEGAL, AND YOU ARE OFFERED INSTRUCTIONS"
2850   PRINT "OF THE GAME, SINCE YOU OBVIOUSLY MISUNDERSTOOD SOMETHING"
2860   PRINT
2870   INPUT "WHEN YOU HAVE READ THIS, STRIKE ´RETURN´",ANSW$
2880   PRINT
2890   PRINT "AFTER YOU HAVE PICKED OUT YOUR COLOUR, THE BANKER WILL ASK YOU TO"
2900   PRINT "BET ON THE GUESS. YOU ARE ONLY ALLOWED TO BET AN AMOUNT OF"
2910   PRINT "DOLLARS. NO CENTS! AND DON´T TRY ANY TRICKS! IF YOU DO, YOU"
2920   PRINT "ARE WARNED BY THE BANKER, AND IF YOU GET FOUR WARNINGS,"
2930   PRINT "YOU ARE MERCILESS TAKEN CARE OF BY THE BOUNCER, WHO WILL"
2940   PRINT "SHOW YOU THE DOOR WITHOUT HESITATION. AND TAKE EXTRA CARE"
2950   PRINT "NOT TO OVERDRAW YOUR ACCOUNT. THAT WILL ONLY BE ACCEPTED ONCE!!!"
2960   PRINT
2970   PRINT "WHEN YOUR BET IS OK, THE WHEEL IS SPUN, AND THE OUTCOME OF THE"
2980   PRINT "GAME WILL APPEAR. IT MAY OF COURSE BE ´BLUE´, ´GREEN´,"
2990   PRINT "´YELLOW´, ´BLACK´, OR ´RED´, AND THE RULES FOR WINNING"
3000   PRINT "ARE THE FOLLOWING:"
3010   PRINT
3020   PRINT "  IF ´BLUE´ WINS, THE BANKER PAYS 1 TO 1 (´EVEN MONEY´)"
3030   PRINT "  IF ´GREEN´ WINS, THE BANKER PAYS 2 TO 1"
3040   PRINT "  IF ´YELLOW´ WINS, THE BANKER PAYS 3 TO 1"
3050   PRINT "  IF ´BLACK´ WINS, THE BANKER PAYS 5 TO 1"
3060   PRINT "  IF ´RED´ WINS, THE BANKER PAYS 11 TO 1"
3070   PRINT "  IF YOU LOOSE, YOUR BET IS SUBTRACTED FROM YOUR ACCOUNT."
3080   PRINT
3090   INPUT "WHEN YOU HAVE FINISHED, STRIKE ´RETURN´",ANSW$
3100 ENDPROC INSTR
3110 REM
3120 REM (*END OF SIMULATOR: CASINO*)
3130 REM //-------------------------------------------------------------//
```

□

A Faire View

BY JEF RASKIN

The first Faire was in the year of the CPU. The second Faire brought in the year of the Floppy. Will next year be the year of the Software?

One thing remains the same: three days of Faire is murder on feet. Last year I went as a reporter and a speaker, this year as an exhibitor. The view was pretty much the same from any of these perspectives—not enough room for all those people. The hall at San Jose, from its ironwood floor to its steel beam ceiling, was smaller than last year's site, and the auxiliary rooms were fewer, if easier to find. Since I belonged in a booth most of the time, I picked up only a few presentations. They were well presented and well attended. In one of them two out of three speakers hadn't shown up, and I ended up filling in with an impromptu lecture.

I can't say much about the hot dogs. I had one, and never went back.

The exhibitors were mostly familiar faces. A few had obviously grown and prospered, some have maintained themselves, others have withered. Some old friends are gone forever.

The packaged, ready-to-fly systems have come a long way. Pet and the TRS-80, which were only rumors and a few handmade prototypes a year ago, are now real. Apple grew from a garage operation to a large, prestigious-image company: you know—stark white and rich teak. Apple even had a disk or two to show (but not to sell). MITS (now Pertec), our progenitor, didn't bother to show up. IMSAI made a weak showing, and seems to be looking at the business market and forgetting the personal computer crowd.

Polymorphic Systems had the neat disk system they've been advertising, and North Star had what looked to be a very similar product. Cromemco, which always has a solid product, had their *quad* disk system with large floppies. There were a lot of companies which proved that they could make a motherboard, a power supply and a box that would hold and power S-100 boards. There were fewer new boards this year—at least fewer radical ones. There seemed to be more memories, and there were more bits per memory board. SWTP kept their line going. Heath was there, and they had floppies too. Everyone had, or was promising, floppies. Most gross thing at the show? A T-shirt, worn by a young woman, which advertised in large letters: 'I have dual floppies.' Funniest thing at show? Jim Warren announcing that the exhibits were to close in five minutes. For the second time. And then, in a faint voice (with the mike still live), 'That was the second time, wasn't it?' Biggest surprise? The good food at the banquet at the Holiday Inn. Best computer? The Terak machine displayed by Dr Bowles of UCSD. Its $7K price tag will keep it out of most of our hands for a while. Neatest packaging? Split it between OAE's EPROM burner and their paper tape reader. Best technical achievement? Apple's extraordinary simplification of the Shugart electronics. Most omnipresent person? John Craig of Kilobaud and his camera. Headiest thing about Faire? Meeting all those people that you usually just read about. They were almost all there.

Gordon French, Adam Osborne, & Jim Warren—Friday banquet

Dave Caulkins, Mike Wilbur, & Ron Crane —PCNET session

A lightly attended exhibit area

Southwest Technical Products Booth

I had a good time with the ALF music synthesizer by joining Carl Helmers at the keyboard for a moment of Mozart.

The PAIA string synthesizer gave me an hour of pleasure as I tried to sound like a Baroque string orchestra. It did a creditable job. I wasn't bad myself.

Last year we were all debating whether the 8080 was better than the 6800 or the 6502 or if the Z80 would rule the world. This year the 16 bitters (sounds like a drink) loom just over the edge of the world. But our sophistication has grown. We now know the utility of hard copy, and there are a rash of clever little printers. We know that we need mass storage, and the Shugart drives spread like Tribbles. This year's software vendors look like some of last year's hardware vendors. Mimeographed sheets, sloppy documentation (with some earning a hearty 'good try there, old chap'), and products scattered like rice at a wedding.

During this coming year software vendors will continue their growth and a few serious personal computing software houses will become prominent in the industry. Meanwhile the manufacturers, having learned for the most part how to manufacture and sell computers, will be trying to learn how to manufacture and sell software. And a few will even turn out some documentation that can be read by people other than the insiders at whom this report is directed.

Apologies in advance to all those exhibitors I haven't mentioned. If this report seems a bit dizzy, it's because that's the way the Faire was. It was every bit a fair, not a convention or scholarly meeting. It was a happy event and a kind of celebration. My compliments to Jim, Bob, and Rick for doing it again. □

An under-attended conference session

Mills College's computer music demonstration

COMPUTERS and the Handicapped

HARDWARE & SOFTWARE HELP SOUGHT in developing a portable communications system (a keyboard and 1-line display) for a person who can't speak.

Steve Gensler
1620 Thousand Oaks Blvd
Berkeley, CA 94702
(415) 524-6162

At the left Robin, a cerebral palsy patient, is shown using her Poly-88 based communications system. The System is controlled by a knee switch; it can be mounted on Robin's wheelchair. Robin can build messages on the CRT either by selecting words from a 1200 word vocabulary or by spelling them out. In our last issue we published the software and necessary hardware modifications for Robin's system.

Robin's system was developed by Tim Scully, shown below at McNeil Island Penintentiary. Tim is now building a similar system for Federal Prison Industries. He is interested in hearing from people working on microcomputer communication systems for the handicapped. His mailing address is Tim Scully, 35267-136 SH, PO Box 1000, Steilacoom, WA 98388.

Stevie Wonder was a surprise visitor to Michigan State University's 'talking computer' center recently, visiting with 'J J' Jackson, a systems analyst at the MSU Artificial Language Laboratory. Wonder came to celebrate the 28th birthday of his friend and former classmate at the Michigan School for the Blind in Lansing. MSU's 'talking computer' has been programmed by John Eulenberg and Morteza Rahimi, professors of computer science, as a speaking—and sometimes singing—voice for handicappers with sight and other physical difficulties.

SKETCHCODE

A Documentation Technique for Computer Hobbyists and Programmers

BY TODD L VOROS

Does the name Todd Voros ring a bell? It should, if you're a fan of good ol' Fortran Man. When not working with Lee Schneider to create more adventures for F-Man, Todd can often be found under the title Systems Software Specialist at A.O. Smith Corporation in New Berlin, Wisconsin.

A computer, in order to perform a useful function, needs to be told what to do. This can be done by a 'canned' pre-written program which one may have purchased, or it can be done by a program which the system user has written himself. If a 'canned' program is unavailable, the user will be forced to write the program himself. The purpose of this article is to illustrate a technique which can help minimize errors in one's programs and help simplify the process of 'debugging' (that is, correcting) a program after it has been written.

The technique is called Sketchcode; it is based on concepts defined in metaprogramming and structured programming. Metaprogramming involves having the user write his program's flow of control in an individualistic, stylistic pseudo-language and then translating that pseudo-language into an actual computer program, which will be executed by his computer system. Structured programming is a programming methodology that helps guide us in defining metaprogramming 'constructs' such as IF-THEN-ELSE, DO WHILE, CASE OF, and defines how they are permitted to be combined. Structured programming has occasionally been called 'GOTO-less' programming. The structured programming concepts contained in this article were first defined by E.W. Dijkstra and Niklaus Wirth.

Programming errors and debugging time are minimized when we have a firm grasp on exactly what it is our program is supposed to be doing when it executes. One well known method of doing this is flowcharting: illustrating the flow of control in a program with a pictorial diagram.

A 'structured program' attempts to illustrate the flow of control in the actual source of the program itself, and the form and syntax of the actual source program (and what the compiler or system interpreter will accept) play an important role in the formation of this type of program. The advantage lies in the fact that when one is debugging the program, the source listing may be used to determine directly the flow of control intended by the author of the program at any given point within the program, without reference to flowcharts.

Unfortunately, structured programming does not lend itself very well to programs written in assembly language because of restrictions imposed upon the programmer by most assemblers. For example, indentation of source statements in a program which utilizes structured programming concepts is often significant and the assembler may not permit this, or lengths of label operands may make writing an indented statement impossible.

The concepts of structured programming and metaprogramming, however, apply regardless of the language being used. Since the aim is to *document the flow of control* within a program, it would be nice if there were some intermediary compromise available. Such a compromise would have to satisfy the requirements of both high-level (BASIC, FORTRAN, etc) users and low-level language (assembler) users. It should help document the flow of control, and be easy to learn and use.

Sketchcode is a metaprogramming pseudo-language intended to satisfy these goals, and is intended to complement, not replace, flowcharting. It is always better to have too much documentation (if such a thing is possible) than too little, especially when a malfunctioning program must be corrected several years after it has been written or when correcting a program you did not write. The use of the metaprogramming philosophy in designing programs can save time and effort when one is coding in *any* computer language. Sketchcode suggests one of the many possible ways in which this philosophy of programming may be utilized.

To get a clear idea of what Sketchcode does, let us first see exactly what programs are made of. A program is an implementation of one or more algorithms intended to solve a problem expressed in a machine digestible form. The algorithms can be composed of processes that do not require decisions and those that do.

Many programmers are familiar with the concept of documenting algorithms by flowcharting: A diamond shape represents a decision, a rectangular box a process not involving a decision. Lines and arrows connect these and other geometric forms together and show the flow of control that takes place when the algorithm is executed by the computer. Sketchcode also has basic components just as flowcharting has. To show the relationship between flowchart representation of an algorithm and its Sketchcode equivalent, the following examples will show both the flowchart and Sketchcode representation of the same logical structure.

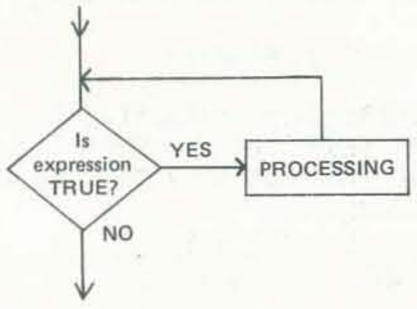
One of the basic ideas in structured programming is that logical levels of control are illustrated by indentation of language statements. Sketchcode, being based on this philosophy, also indents statements. One of the reasons we indent statements is to show where the majority of a program's execution time is spent, and under what conditions certain sections of code can be executed.

Most computer programs have loops. A *loop* can be expressed in Sketchcode as follows:

```
DO WHILE (an expression);
  PROCESSING
END;
```

Note that PROCESSING is indented two spaces to the right. All other sketchcode processing within that loop will be indented two spaces to the right.

Here's a flow chart for a loop:



The (an expression) part of the loop may contain any number of variables; the evaluation of the expression results in the assignment of a TRUE or a FALSE condition. While the condition remains true, we will execute statements contained inside the loop. If the condition is false, we do not execute any statements in the loop; it is as if we had 'fallen through' the loop without stopping to examine anything within it. We will simply begin executing statements after the END; which signals us where the loop ends. This is why it is not indented two columns to the right like PROCESSING.

We can get out of a Sketchcode DO loop by having PROCESSING within the loop alter the value of one or more variables contained within (an expression). For example, to execute some process 10 times we can write:

```
COUNT = 1
DO WHILE (COUNT less than 11);
  PROCESS
  COUNT = COUNT + 1
END;
```

Naturally, the expression that is tested for TRUE or FALSE could be much more complex, e.g. DO WHILE (I = A+2 OR B = C–D). In addition, we can put a loop within a loop, always making sure to indent two spaces to the right when appropriate:

```
DO WHILE (I less than 10);
  PROCESS
  DO WHILE (J less than 5);
    MORE PROCESSING
  END;
END;
```

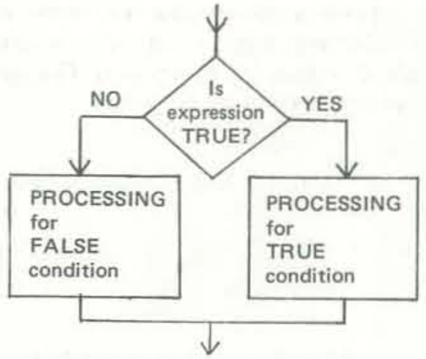
and observe that each DO has its own closing END statement.

This way of representing the logical flow of control of a program allows you to clearly and concisely express some fairly complex situations involving loops. Note that the *inner* DO loop was indented two columns to the right and processing performed under *its* control was itself indented two columns to the right. Thus the deeper a loop (ie the more nested it is in the logical flow of control of the program), the further to the right it will appear in the program's Sketchcode representation. Code that is indented farthest to the right will also probably be executed more often than other portions of the program, so if you have written a Sketchcode representation of your program you should concentrate any optimizing efforts on innermost loops *first*. However, programs are not composed just of loops, and we must consider other elements of a computer program.

Decisions are also of prime importance in directing the flow of control within a program. In Sketchcode, a decision is always represented by a structure of the following form:

```
IF (expression)
  THEN DO;
    PROCESSING performed if expression
      is TRUE
  ELSE;
    PROCESSING performed if expression
      is FALSE
```

which is how Sketchcode implements the IF-THEN-ELSE construct. The flowchart equivalent is:

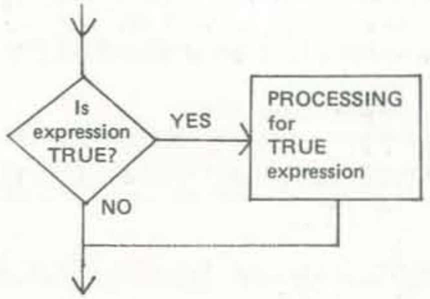


Notice that for readability the THEN DO; and the ELSE; are indented two columns to the right and their corresponding processing is itself indented two columns to the right. Since a Sketchcode expression is required to be TRUE or FALSE *either* the processing under the THEN DO; will be executed and the processing under the ELSE; will be skipped, *or* the processing under the THEN DO; will be ignored and the processing under the ELSE; will be executed.

There exist two special cases of the Sketchcode constructs discussed so far. These are when we wish to do nothing based on some condition and when we wish to do something forever (a never-ending loop). We can 'do nothing' in an IF-THEN-ELSE if we omit the ELSE; which permits us to execute some processing *only* if some condition defined by (an expression) is true and to do nothing otherwise:

```
IF (an expression)
  THEN DO;
    PROCESSING
OTHER SKETCHCODE STATEMENTS
```

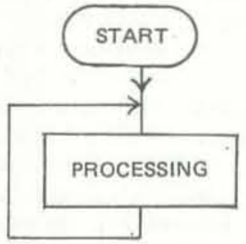
Here's the equivalent flowchart:



and to solve the problem of the never-ending loop we introduce the Sketchcode word FOREVER:

```
DO FOREVER;
  PROCESS
END;
```

And the flowchart:



An example where we may wish to employ the DO FOREVER construct is in documenting a program which once loaded into our machine will request input from the user, process it, print out a result, and await further input in a never-ending cycle.

Finally, Sketchcode allows for the use of subroutines. A subroutine in Sketchcode representation is *invoked* by a CALL statement. A subroutine in Sketchcode is *defined* by giving the subroutine a name followed by a colon, and indenting all statements in that subroutine two columns to the right under the label. A subroutine ends with a RETURN; statement. The RETURN; statement aligns with the label giving a name to the subroutine. In Sketchcode only *one* RETURN; may appear in a subroutine. A subroutine may have *one and only one* entry point and *one and only one* exit point. This may seem to be a severe restriction but it will *enforce* a top-to-bottom flow of control within a subroutine. For example, to invoke a subroutine to return the larger of two numbers we could write:

```
CALL BIGGER (A,B,BIGGEST)
```

and at some point define BIGGER:

```
BIGGER: (A,B, BIGGEST)
  BIGGEST = A
  IF (B IS GREATER THAN A)
    THEN DO;
      BIGGEST = B
RETURN
```

In Sketchcode you can either assume all variables are known to all subroutines (all variables are *global*) or you can pass variables to a subroutine explicitly by putting them in a list enclosed within parentheses after the call statement and having a corresponding list after the label defining the name of a subroutine. It is a good idea when writing Sketchcode subroutines to start them on a fresh piece of paper rather than mix them in with other Sketchcode.

The structures we have defined are *completely* adequate for the expression of any problem capable of implementation on a home hobbyist computer system. But, you may ask, 'WHERE ARE THE GOTO STATEMENTS?' (or jumps, or branches if you prefer). The answer is there *aren't* any in Sketchcode. Program logic *always* flows from top to bottom, through various levels of indentation on the way, and program loops are *always* clearly documented. Sketchcode forces you to provide a clear, concise definition of what you wish your program to do, but still allows you to express yourself in an individualistic style. (Our own examples certainly aren't part of any 'legal' programming language.) When you have written your program's logic in Sketchcode you will find it easier to follow for both yourself and others and if you have defined the *logic* (not the actual *coding* of your program) you can write your program for a different computer with much less effort. And last, but not least, if you *really* want to make your programs self-documenting, *include* your Sketchcode representation of the program as part of the COMMENTS in the assembly language version of your program (show what each Sketchcode statement expands into in actual machine instructions). However, no matter what the language, Sketchcode should assist you in providing better documentation and insight into your program's operation.

hints

Here are a few hints on the use and writing of Sketchcode based on two years of working with it:

- If you find yourself writing a lot of IF-THEN-ELSE, IF-THEN-ELSE clustered closely together in your Sketchcode, ask yourself the question: *'Is this really a DO in disguise?'*
- Remember that searches through tables, lists and arrays are usually implemented by DO's.
- Don't forget to indent when going to a deeper level of control.
- Remember that all IF's do not necessarily require an ELSE!
- If possible, break up large numbers of sequential Sketchcode statements into subroutines. Try to make a subroutine fit on one page of paper, if possible, decomposing it into two or more deeper subroutines if necessary.
  Example:

```
INITIALIZE        A:                B:
CALL A              CALL A1           PROCESS
CALL B              PROCESS          RETURN
CALL C              CALL A2
STOP              RETURN

Page 1            Page 2            Page 3
```

and so forth. . .

Processing performed under the legs of an IF (the THEN DO and ELSE) can be switched by negating the results of the expression you're testing. Thus,

```
Thus, IF (X=0)
        THEN DO;
          A=B
        ELSE;
          A=B+B
```

is the same as

```
IF (X not equal to 0)
  THEN DO;
    A=B+B
  ELSE;
    A=B
```

The following point is somewhat tricky, but worth consideration if your Sketchcode doesn't 'seem right': If the ELSE condition of the IF can be reached by code *prior* to the IF test, then it is *not* an ELSE condition. Remove the ELSE and the indentation of the code under the ELSE.

Ask others to review the Sketchcode representation of your program. This can help detect errors you have not caught yet.

Before you begin to write down the very first machine or assembly language statement of your program, have the *completed* Sketchcode representation of your program in front of you and code your actual program from the Sketchcode directly.

On the opposite page we demonstrate how the *same* Sketchcode listing (the metaprogram representation of the solution to the problem) can be used to document programs for two quite different machines, one in 8080 code and the other in 6502 assembly code.

Here's an example of how Sketchcode can be applied to solve a problem.

The problem: Determine the largest and smallest numbers stored in an array in memory. The smallest number possible will be zero, the largest will be the maximum the machine can represent.

Assumptions:

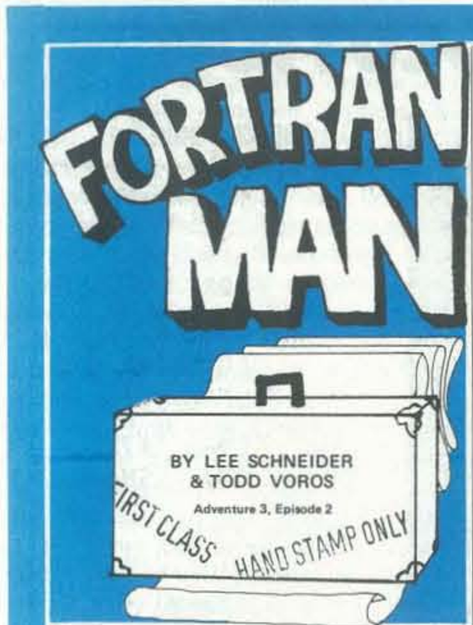
1. Array is in sequential location (not scattered over memory).
2. The array consists of N elements.
   ARRAY(1) retrieves the first element of the array.
   ARRAY(N) retrieves the last element of the array.
   ARRAY(5) retrieves the fifth element of the array, and so forth.
3. INITIALIZE sets up our machine specific environment necessary to operate. (Clears registers, for example.)

□

A SKETCHCODE SOLUTION

```
         8080 CODE                   SKETCHCODE                          6502 CODE

          ORG   1ØØØH
INDEX:    EQU   Ø
          LXI   SP,STACK    INITIALIZE                                  LDX   #STACK
                                                                        TXS
          MVI   A,Ø         BIG=Ø                                       LDA   #Ø
          STA   BIG                                                     STA   BIG
          MVI   A,7FH       SMALL=BIGGEST NUMBER                        LDA   #7F
          STA   SMALL                                                   STA   SMALL
          MVI   A,5Ø        ARRAYSIZE = SIZE OF ARRAY
          STA   ASIZE
          MVI   INDEX,Ø     INDEX=1                                     LDX=Ø
          MVI   C,1
DO:       LDA   ASIZE       DO WHILE (INDEX<=ARRAYSIZE),        DO:     CPX   #ASIZE
          CMP   C                                                       BEQ   ENDDO
          JC    ENDDO
          LHLD  AADDR        IF (ARRAY(INDEX)<SMALL)                    LDA   ARRAY,X
          DAD   INDEX                                                   CMP   SMALL
          MOV   E,M
          LDA   SMALL
          CMP   E             THEN DO,                                  BPL   T1
          JC    T1
          JZ    T1
          MOV   A,E            SMALL=ARRAY(INDEX)                       STA   SMALL
          STA   SMALL
T1:       LDA   BIG          IF (ARRAY(INDEX)>BIG)              T1:     CMP   BIG
          CMP   E             THEN DO,                                  BMI   T2
          JNC   T2
          MOV   A,E            BIG=ARRAY(INDEX)                         STA   BIG
          STA   BIG
T2:       INX   INDEX        INDEX=INDEX+1                      T2:     INX
          JMP   DO          END DO,                                     JMP   DO
ENDDO:    CALL  PRINT       PRINT BIG,SMALL                     ENDDO:  JSR
          DW    BIG
          DW    SMALL
HALT:     HLT               STOP                                HALT:   JMP   HALT
          JMP   HALT
BIG:      DS    1
SMALL:    DS    1
ASIZE:    DS    1
AADDR:    DW    ARRAY-1
ARRAY:    DS    5Ø
          DS    1ØØ
STACK:    DS    ØH
          END
```

FORTRAN MAN



BY LEE SCHNEIDER & TODD VOROS

Adventure 3, Episode 2

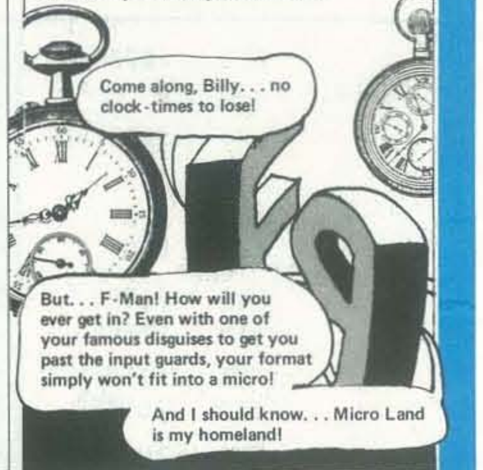
In our last episode, Our Hero had just returned from his visit to the 'Old Country' of Transistoria. Finding a little 4-legged micro-beastie running about his resident location, F-Man attempts to communicate with it. . . but the differences in their coding make direct translation impossible!

F-Man CALLs in his young partner in crime-fighting, Billy Basic, to act as interpreter. . . and with Billy's aid he is at last able to read out the message, which has been cleverly concealed in a ROM built into the micro-beastie.

The message comes from Microprocessor Land—otherwise known as the Land of the Little People—where a nefarious villain calling himself the Glitchmaster has stolen the Lockout Monster from its rightful owners in Clan McIntel. With the monster under his control, the Glitchmaster has managed to cut off all I/O from the land, and is even now settling down to rule it!

The Underground Resistance Movement, led by General Wirewound, is attempting valiantly to overcome the Glitchmaster, but it is a losing battle. They have put out a call for help. . . a CALL which Fortran Man cannot ignore!

Wasting little time on null cycles, F-Man branches out of his resident location and vectors himself immediately towards the downtown area of 360 City, intending to relocate himself immediately to Microprocessor Land!


Come along, Billy. . . no clock-times to lose!
But. . . F-Man! How will you ever get in? Even with one of your famous disguises to get you past the input guards, your format simply won't fit into a micro!
And I should know. . . Micro Land is my homeland!

But as usual, Fortran Man has already come up with the solution. . . and as they branch hurriedly through the mass storage areas of 360 City, he explains to Billy. . .

Don't Worry, Billy. . . I have already computed that problem!
I have arranged with Firmware Interface Control to have myself re-coded and transferred onto a set of unmarked PROMs. . .
And then you, my friend, shall simply carry me onto the bus and transport me into Micro Land!

Together they enter Firmware Interface Control, and a mere hundred programming loops and a Verify later, Billy emerges with a plain, unmarked chip carrier under his arm. . .

Once there, you will take me directly to the facilities run by Clan McIntel. . . and there, through a simple bootstrap routine included in the PROM, I shall be reconstructed!
Amazing, F-Man!
A very basic trick, Billy! Now remember. . .
FIC
Heavens to Coding Form! Carrying the one and only Fortran Man under one arm sure makes me nervous. . . sure hope I don't drop any bits!
Well, now down to the bus. . .

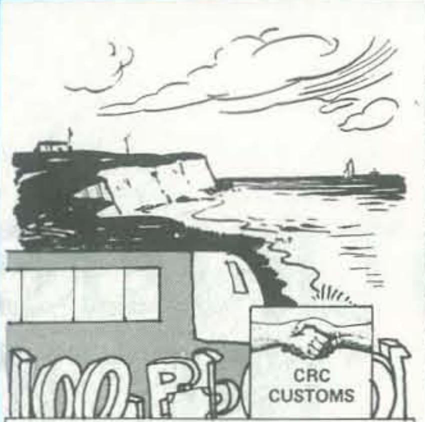
BUS LOADING NOW TAKING PLACE FOR ANALOGVILLE, DMA STATION AND MICROPROCESSOR LAND. ALL ABOARD!
I sure hope this works. . . otherwise my poor homeland is in serious trouble!

Branching towards the peripheral edges of 360 City, Billy Basic lines up amongst the rest of the bit-stream awaiting an opportunity to get onto the bus. . .

Once aboard the bus, the load is automatically balanced to avoid bus oscillation or overloading. Then the signal is given; Billy and Our Hero (safely in PROM storage) are gated out of 360-City towards the place where they are needed.

488
Micro Land 500 µsec
CHECK YOUR CLOCK SPEED

It is not a short trip, but the high-speed bidirectional bus takes considerably less time than the low-speed channels which run to other peripheral lands. After a number of brief stops for data exchange at various bus connection points, they at last approach the Land of the Little People. . .



At Micro Land the bus line is terminated, and all data is unloaded. Billy joins the lines awaiting the usual check through the interface logic before being allowed to enter. . . noting that the I/O channel also ends here, with data from both sources being shared through one interface terminal. . .

At last he reaches the Customs' Gates. . . As usual, security is very evident everywhere, and especially so now that the Glitchmaster is loose!

CUSTOMS
NEXT!
All right; luggage inspection! On the data table, please!

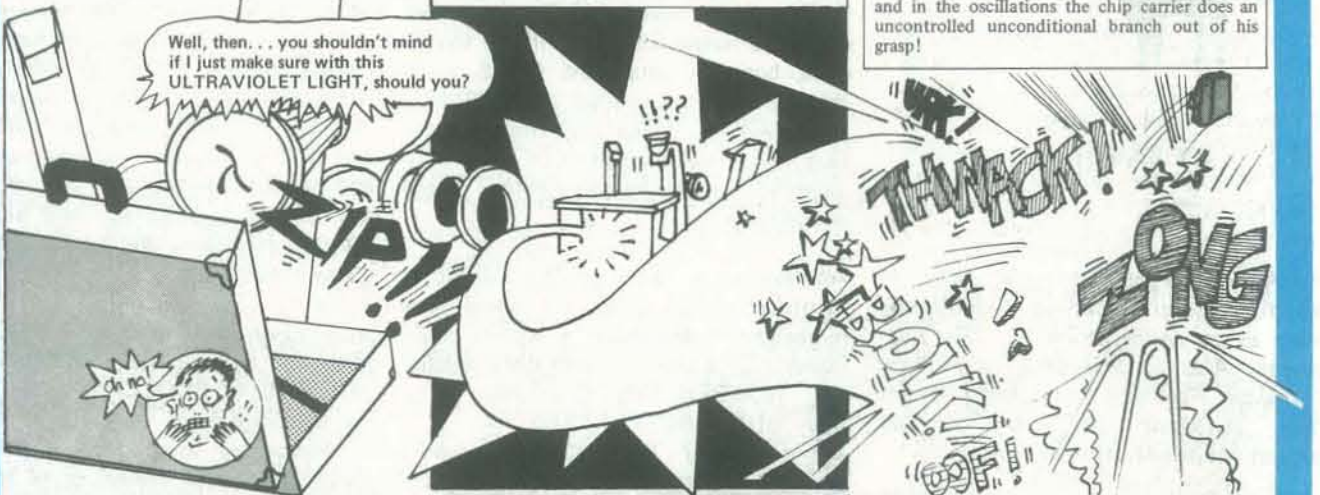
Well, now. . . what have we here?
Er. . . just some souvenier PROMs for my collection!
You're sure they are empty?
Why. . . um. . . of course they are!

Reluctantly, Billy complies with the order. . . noting that these are not the usual customs agents, but members of the Elite Data Security Guard. . . and little escapes *their* notice. . .

The inspector looks thoughtful for a moment. . . then, before Billy can execute another statement. . .

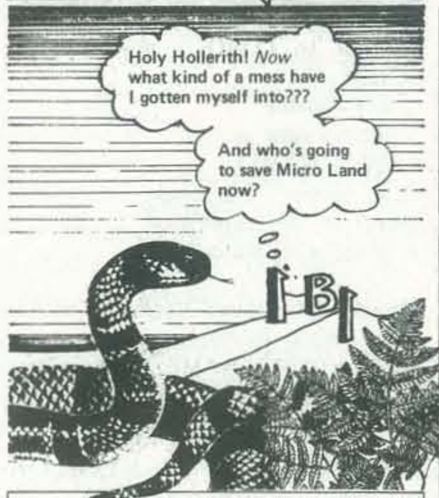
Seeing that Our Hero is in imminent danger of being wiped out and lost forever, Billy executes the only option possible. . . he grabs the chip carrier, turns, and goes into RUN state!

But try as he might, little Billy Basic cannot outrun the faster-executing data security routines. . . and within a few cycles he is descended upon by the guards! They struggle. . . and in the oscillations the chip carrier does an uncontrolled unconditional branch out of his grasp!


Well, then. . . you shouldn't mind if I just make sure with this ULTRAVIOLET LIGHT, should you?
ZIP!
Oh no!
!!??
URK!
THWACK!
ZONG
BONK!
OOF!

And moments later, poor Billy is under their power, held firmly by the control lines. . .

Here he is, Commander! The case he carried fell into the data channel. We could not find it!
No matter!
So! Another data smuggler, eh? We have ways of dealing with your kind. . . your transmitting days are over! You'll be placed in HOLD state for the rest of your natural cycles!
Take him away!
sigh.

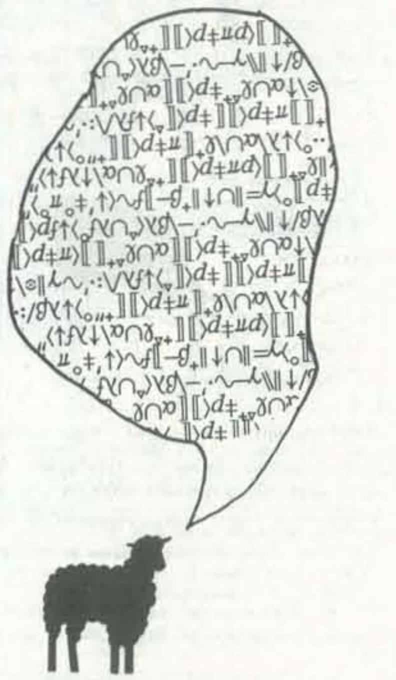

Holy Hollerith! Now what kind of a mess have I gotten myself into???
And who's going to save Micro Land now?

And so, firmly in custody with his line numbers taken away, Billy Basic is led slowly towards the HOLDing center. . . and a dark gloom settles over Microprocessor Land. . .

And somewhere, floating gently in the third state of the channel, a little chip carrier ripples slowly down the data stream with the currents. . .

Is this the final END for Fortran Man? Will Billy Basic ever be free to RUN again? Will the Glitchmaster triumph in his domination of Micro Land?

Tune in again next issue. . . same time, same memory address!



TINY LANGUAGE TALK

BY SAM HILLS

Reader Hills' suggestions are of sufficient length and depth that we're publishing them as an entire article on Tiny Languages. Also in this issue is a 'Tiny Language Feedback' in which readers raise questions and comment on suggestions previously published.

Here are some suggestions for your new games language.

1. VARIABLE NAMES

One of the most serious shortcomings of BASIC is its one- and two-character variable names. Actual experience with a variety of languages has shown that 8 characters is the absolute minimum for readable programs, and sometimes even more would be helpful. For example: which of the following would you prefer to see in a program you were trying to understand:

```
     S = S+N
or   SCORE:= SCORE+NEW_PIECES
     ??????
```

Some languages go overboard in allowing long variable names (such as COBOL, which allows up to 30 characters in a variable name), and this eats up valuable memory space in a hurry. The best suggestion I have seen is to allow unlimited-length identifiers in the source code, but only retain the first 8 or 10 characters in the symbol table. (This is what PASCAL does.) This allows identifiers which are descriptive of what they identify, yet it keeps memory usage reasonable.

Another point while on the subject of identifiers: be sure to allow for hyphenated identifiers! I would *much* rather read a program with the identifier

NUMBER_OF_PLAYERS

than NUMBEROFPLAYERS

wouldn't you? The PASCAL compiler which I am currently using allows the underscore as the hyphen; this is far superior to COBOL's minus sign! (In COBOL:

MY–NAME

is *not* the same as

MY – NAME!

The former is a single identifier, the latter is an expression involving the subtraction operator!)

Should consecutive hyphens, or identifiers ending in hyphens, be permitted? (Sure, YOUR__NAME_ looks awkward, but it takes extra code in the scanner to trap it, and it doesn't really hurt anything.)

2. LINE NUMBERS

Line numbers have no place in the programming language—they should be used *ONLY* to specify which line to edit when editing the program!!!!!

3. STANDARD TYPES

Limiting the language to strings places an unnecessary burden on the interpreter when doing arithmetic. You need numeric variables too. Whether to have both INTEGER and REAL or simply type NUMERIC should be up to the programmer; let the younger kids use NUMERIC, and after they learn more about numbers and begin to write bigger programs they can advance to INTEGER and REAL, in order to save execution time and perhaps memory space too.

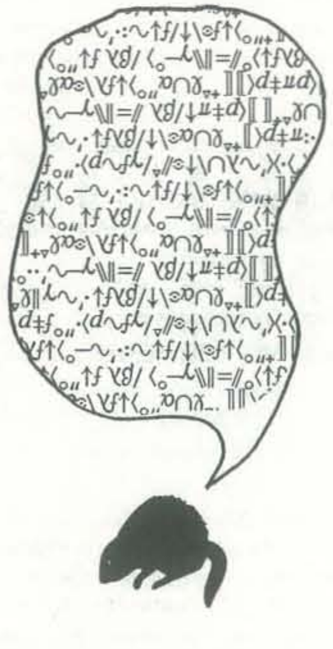
In addition, you need type BOOLEAN (although I prefer the FORTRAN name LOGICAL to BOOLEAN—more people know about logic than about Boole!—or maybe we should call it BINARY; that's what it is, really) so as to avoid the absurdity of having to assign either of two numbers to a variable, when we really wanted to express a TRUE/FALSE condition.

Since the language will be used to draw pictures, we need the standard type COLOR. With a black and white TVT, COLOR would be defined as TYPE COLOR = (BLACK, WHITE);. For a color CRT, such as the Cromemco DAZZLER or the COMPUCOLOR, we could define:

```
TYPE COLOR = (BLACK, RED,
  BLUE, GREEN, YELLOW,
  MAGENTA, CYAN, WHITE)
```

(Some people may object to MAGENTA and CYAN; they could substitute the less accurate names VIOLET and AQUA or PURPLE and BLUE_GREEN.)

Naturally, 'arithmetic' could be performed on colors:



```
RED + GREEN→YELLOW
MAGENTA – RED→BLUE
YELLOW + BLUE→WHITE
GREEN – GREEN→BLACK
etc.
```

In a system where colors may have several different intensities, the color constants would represent fully saturated colors, while pastel tints could be produced by multiplying the appropriate constant by a number between 0 and 1. For instance, RED + 0.5 * CYAN would produce pink, while RED + 0.5 * GREEN would produce orange! (Note that the above system of performing 'arithmetic' on colors is a good way to teach kids about primary and secondary colors, and how they mix!)

4. CONTROL STRUCTURES

The absolute minimum set of control structures required is:

```
LOOP
  :

EXIT IF ...
  :

REPEAT;
```

(This will adequately take the place of WHILE and UNTIL control structures, although you may want to include them anyway.)

```
FOR ... STEP stepsize ... NEXT;
```

(STEP is optional, assumed to be +1 if omitted.)

```
IF...THEN        or      IF .. THEN
  :                        .
                           .
ELSE                     ENDIF;
  :

ENDIF;
```

(ENDIF is necessary to show where the predicate ends. The other alternative is to require the predicate to be enclosed in BEGIN ... END brackets. Personally, I prefer ENDIF.)

```
CASE expression OF
expression:  statements;
expression:  statements;
  :

ENDCASE;
```

Here again, ENDCASE is needed to show where the last limb of the case ends. You may argue that the CASE statement isn't really necessary. It isn't, really, but it sure *is* a lot easier to understand than an awkward string of ELSE IF's!

The syntax of the CASE statement must allow for an OTHERS limb, to prevent cluttering up the program with an IF to test whether or not the case expression can be satisfied. A point of discussion: what should happen if none of the case labels satisfies the case, and there isn't an OTHERS label? Should the program just continue with the first statement after ENDCASE, or should this cause an error condition?

Please note that the above list of control structures does *NOT* contain a single GOTO!!! With the above set of control structures, GOTO's not only aren't needed, they actually hurt matters!!!!!

The above control structures may be nested to any depth desired, in any combination desired. (A practical limit might be until the stack starts to grow down and eat into the data area or some other constraint based on memory limits.)

The above control structures are adequate to express *any* algorithm, no matter how complex. And what's more, they force one to *think* algorithmically!

5. PROCEDURE (SUBROUTINE) CALLS

Procedures must be callable by name, and *must* include the ability to pass parameters. (This is one of BASIC's most serious shortcomings.) Arguments should be checked for type compatibility, preferably *before* execution begins. This would be simple in a compiler; in an interpreter it would require a pre-execution error-checking scan, which would be a good idea anyway—a lot of obscure errors (the kind that don't always show up every time the program is run) could be detected this way.

The language should also allow for procedures defined external to the program. This would include pre-defined functions such as RANDOM (which returns a number between 0 and its argument) and SQUARE_ROOT (guess what this one does?). ANSWER prints its (STRING) argument on the terminal and waits for a 'yes' or 'no' answer, (looping and re-prompting if anything else is typed!), and returns the BOOLEAN value TRUE or FALSE depending on the answer). We'll also want to permit user-defined subroutines and functions. The latter would be stored on disks in systems which have disks, or could be loaded from tape when disks were not available.

I imagine a typical dialogue between the user and the monitor in a tape-based environment would go something like this:

NEW (Resets the load pointer to the beginning of the free memory.)

LOAD (User now loads tape containing a subroutine he wrote last week. This program segment is loaded into the space *following* the previous one because he didn't give a NEW command.)

LOAD (Another subroutine, etc.)

RUN (Check the program for missing procedures, incompatible argument types, undeclared identifiers, etc, and if everything's OK, execute the program.)

SAVE (Save the *whole* program, including the subroutines loaded from the second, third,. . .tapes on *one* tape. (This'll save a lot of time on subsequent loads!))

```
SAVE DRAW_SQUARE,
  DRAW_CIRCLE
          (This command saves only the
          subroutines DRAW_SQUARE
          and DRAW_CIRCLE on tape
          (so they can be loaded onto
          the end of another program,
          if desired.).)
```

Subroutines SAVEd with a single ŞAVE command would be saved as a single file; subroutines saved with separate SAVE commands would be saved as separate files, just like when you save several BASIC programs on the same tape.

6. LOCAL VARIABLES AND SUBROUTINES

These are virtually a necessity with the above: can you imagine the difficulty of having to check all of the pre-recorded subroutines that you intend to use to make sure you haven't used any identifier in one of them that you want to use in your main program????

7. RECURSION

Of course we want recursion! And it's not at all difficult to implement on any processor which uses a stack.

8. COMMENTS

We must include a method for putting comments into a source program. The method I like best is the one used by PASCAL: everything enclosed between 'comment braces' (the symbols (* and *) in PASCAL) is considered to be a comment, and is ignored by the compiler or interpreter, no matter where it appears in the statement. For example:

```
GAME† OVER
```

9. INITIALIZATIONS & CONSTANTS

I have *never* seen a *worse* initialization scheme than BASIC's READ. . DATA construct. FORTRAN's DATA statement is better, but PASCAL's CONST declaration is the best yet. In PASCAL, one can say:

```
CONST (* declare constants *)
  LINE_LENGTH: = 80;
  GUESS_MAX: = 10 (* limit of 10
    guesses per player *);
```

In some versions of PASCAL (such as the DEC-10 version), one can also have an INITPROCEDURE which initializes variables:

```
INITPROCEDURE;
  BEGIN
    BOARD [0 . . 8, 0 . . 8] := 0;
  END;
```

This acts just like the FORTRAN statement:

```
DATA BOARD /64 * 0./
```

10. MORE STANDARD TYPES

In more advanced versions of the language, you may want to copy some of PASCAL's other standard types: SET, RECORD, POINTER and SCALAR. Sure, the little kids won't know what to do with these concepts, but who says that a kid has to learn the WHOLE language in his first lesson??? For instance, in all the FORTRAN programs I have written, I have never needed the COMPLEX or DOUBLE PRECISION data types, except for a couple of rather trivial class assignments. However, my programs were never adversely affected by the fact that those types were available if needed. By including these more advanced types in the language definition, older kids (and adult game-writers, too) can use them when they learn how. Actually, the standard type TURTLE is just a special type RECORD, and the standard type COLOR is just a special type of SCALAR!

11. SUBRANGES OF ARRAYS

It would be nice to allow a subrange of an array to be assigned into a subrange of another array, rather than just requiring whole arrays to be assigned. For example, given the following declarations:

```
ABC: ARRAY (0 . . 10) OF INTEGER;
DEF: ARRAY (0 . .10) OF INTEGER;
```

one could obviously write

```
ABC: = DEF;
```

to copy the entire contents of array DEF into array ABC. It would also be nice if it were possible to write

```
ABC (3. . 5) : = DEF (7 . 9);
```

to copy elements 7 thru 9 of DEF into elements 3 thru 5 of ABC.

12. CONCATENATION OF STRINGS

Obviously, we need a way to concatenate strings. But do we allow

```
STORY; = STORY + 'The end.';
```

or do we use a standard procedure to do this?

```
STORY: = JOIN (STORY, 'The end.');
```

Whichever method is chosen should be consistent with the method used to implement substrings.

13. INPUT ERROR RECOVERY

Nothing is more maddening than to have a program crash because you typed a non-numeric character to a numeric-input routine! Our input routines must be written to check for the right kind of input, and, on an error, to simply re-prompt and try again.

A typical dialogue might go like this, with the computer typing in upper case, the user in lower.

```
DO YOU WANT TO PLAY AGAIN?
sure!
SORRY, I DON'T UNDERSTAND
  SURE! PLEASE ANSWER YES
  OR NO.
DO YOU WANT TO PLAY AGAIN?
yes
HOW MANY PLAYERS?
two
SORRY, I NEED A NUMBER.
HOW MANY PLAYERS?
2
```

In systems where memory is limited, you may want to eliminate the SORRY, . . . message, and merely repeat the prompt, however, under *NO* conditions whatsoever, should *ANY* possible input cause the program to crash with a message like

```
ERROR 25 IN LINE 645; INVALID
  INPUT STRING TO NUMERIC
  INPUT
```

This will require the input function to check what type of response is required, and generate the appropriate re-prompt when needed. One alternative would be to have 3 separate input functions:

```
FUNCTION ANSWER: BOOLEAN;
  (* only accepts YES or NO *)
FUNCTION GET_NUMBER:
  NUMERIC; (* only accepts
  numbers *)
FUNCTION INPUT: STRING;
  (* accepts any string *)
```

All 3 of these functions would accept a STRING argument which is printed as the prompt, much like the INPUT of BASIC.

The other alternative would be to have only *one* standard procedure INPUT, in which case the interpreter would have to check to see what type of variable the result was being assigned to (BOOLEAN, NUMERIC or STRING). This latter approach, while (maybe) being a little bit harder to implement would be much easier for young programmers to grasp.

14. AUTOMATIC STRING/NUMERIC CONVERSION

Perhaps we should include automatic string/numeric conversion, just like we have automatic integer/real conversion. For example, suppose NUM were declared NUMERIC and STR were declared STRING: It should be OK to write

```
STR: = 'The answer is ' + NUM; and
NUM: = 25 – STR;
```

In the latter case, STR *must* contain the string representation of a number. (If not, what should happen? Should this be a run-time error, or should the string-to-numeric conversion routine ignore anything that isn't a digit? Or should we just forget the whole matter and only allow NUMERIC to STRING conversion???)

15. DECLARATION OF VARIABLES

Should we require all variables to be declared, or only those special types like COLOR or arrays? I admit it gets to be a hassle to have to declare every variable, but it sure catches a lot of misspellings! I'd much rather have to declare everything than to have the compiler generate a new variable on account of a spelling error in a section of code which gets executed only once in a blue moon!

16. PRETTY OUTPUT

Of course, our interpreter should have a 'prettyprinter' to format the source code to highlight the block structure. (See any of Mac Oglesby's game listings for an example of this.) Such a program would make it very easy to spot errors in block structure; for example, a missing ENDIF or NEXT would cause the listing to fail to close up the left margin at the final END.

An easy way to implement this in the internal representation of the object code without taking up a lot of memory space with blanks is as follows:

first two bytes: line number (for EDITING purposes *ONLY*!!!)

3rd byte: length of this line (i.e., a pointer to the beginning of the next line)

4th byte: number of spaces to indent this line in the source listing;

5th thru nth byte: the actual source line,

(n + 1) th byte; carriage return.

This format requires *much* less memory than storing all those leading blanks, *and* it makes life *much* easier for the formatter, because it now has only to change the indentation count, rather than actually add or remove spaces every time the program is edited. With this system, leading blanks in the source statements would be stripped off before the lines were stored, so as not to waste memory space (and cause extra work for the formatter).

17. MINIMUM EDITING STANDARDS

No one should ever be forced to retype an entire line just to change one letter! The absolute minimum set of editing commands should contain the following:

Insert a new line.

Delete a line.

Delete a line, and replace it with a new line.

Move a line, or group of lines, to a new spot in the program.

Split a line into two lines.

Step thru a line, one character at a time, in either direction.

Insert and delete single characters from within a line.

LINED An editor on the DEC-10 uses ctrl-A step forward thru a line, and RUBOUT to delete characters. Insertion of characters is accomplished simply by typing the new characters in where desired. (Ctrl-A can be used to reach the desired point to insert.) LINED also has some more sophisticated features, such as ctrl-S to step thru a line until a specified character is found, and ctrl-F to search for a specified *string* of characters. However, these features aren't really absolutely necessary in a minimal-features editor. I also suggest the use of BACKSPACE to step *backwards* thru a line; LINED doesn't have this, but it would sure be nice to have, and would only require maybe a half-dozen bytes of code to implement.

Also needed is a control character to step thru the entire remainder of the line when no further changes are to be made on that line. LINED uses ctrl-E to step to the end of the line and stop (for instance,

to add something on to the end of the line), and ctrl-R to release the remainder of the line (i.e., no further changes to be made to the current line).

And we need a way to insert blank lines into the listing to separate code dealing with separate parts of the problem—e.g., the input routines, the initialization routines, the computational routines, the graphic routines, etc. One possible way of doing this: typing a line number, followed by a carriage return inserts a blank line, while typing D followed by a line number deletes the specified line.

18. STRING-TO-NUMERIC CONVERSION

What happens when you try to add '25' + '25'? Does it convert the two strings to numbers, yielding 50? Or does it concatenate the strings, yielding '2525'? This is a problem only if you use the + sign to denote string concatenation, and you allow automatic string-to-numeric conversion. One way out of this dilemma would be not to allow automatic string-to-numeric conversion, but to provide the standard function VALUE:

```
    VALUE ('25') → 25
```

The other alternative would be to require the standard function JOIN be used to concatenate strings.

19. STANDARD FUNCTIONS AND PROCEDURES

Several standard functions/procedures have been mentioned already; RANDOM, INPUT, ANSWER, SQUARE_ROOT, VALUE, DRAW, TURN, etc. We'll also need the trig functions SIN and COS. Do we really need TAN? After all, TAN(X): : = SIN(X)/COS(X). And we can also construct ARCSIN and ARCCOS. We'll need ASCII (which returns the ASCII value of its string argument) and CHR (returns the string specified by its ASCII-valued argument). We'll need EXP and LOG too. (The little kids won't understand it, but we should have them for the older kids and adult game-writers.) And INT, MOD, SIGN and ABS.

If we have the standard type COLOR, we'll probably need some way to convert strings to colors, and vice-versa. If the INPUT function is required to return the value of a color, it can loop and reprompt, just like with any other invalid input, but what if string-to-color conversion occurs somewhere else in the program? For instance, if a string argument is supplied to the standard procedure DRAW?

20. DYNAMIC TYPES

Snobol and some other languages allow dynamic types: the type of a variable is determined by the last value assigned to it. This would eliminate the need to predeclare all variables (except arrays—we still have to specify the number of subscripts, and their upper and lower bounds). But it would lead to a lot of errors not being detected until run-time that could be detected at compile-time otherwise. Consider this example: using a variable containing a COLOR as the condition in an IF.

```
    XYZ: = RED;
    IF XYZ THEN . . . .
```

The above situation would cause a run-time error with dynamic-variable types, and a compile-time error with predeclared types.

And should the type of a dynamic variable be changeable once it has been assigned, or should it remain the same for the remainder of execution? The latter would reduce the tendency to use the same variable for two different, unrelated things in different parts of the same program.

21. STRING MATCHES AND SUBSTITUTIONS

Of course, we'll want some sort of string matching test—probably a BOOLEAN function which will work analogous to PILOT's M command: the first (string) argument is compared with the parts of the second argument, looking for a match. But, unlike PILOT, we should supply the test string in an argument, rather than limit ourselves to the latest input string—this permits much greater flexibility in the programs. We'll also want a string substitution procedure: search a target string for a given substring, and replace it with a given replacement substring. These are both quite useful in a conversational-type environment.

22. CALCULATOR MODE

When in Calculator mode, the keyboard should respond EXACTLY like a calculator, not like BASIC which requires a leading PRINT rather than a trailing equals sign to do immediate calculations!

23. JOYSTICKS

In any language used for games, we'll want a way to input the position of joysticks. This could be done with a standard procedure which takes 3 parameters: the first would be the number of the joystick, and the other two would return its position. Alternately, if RECORD is a permitted type, the JOYSTICK routine could be a FUNCTION, returning as its value a record containing the X and Y coordinates of the joystick. Yet another alternative would be to number the joysticks by even numbers, so JOYSTICK (2*N) would return the X-coordinate of joystick N, and JOYSTICK (2*N+1) would return its Y-coordinate.

A simple routine is shown below which would allow a child to use a joystick to guide a turtle over the CRT screen:

```
    LOOP;
    PLOT (WHITE, JOYSTICK (1),
        JOYSTICK (2));
    REPEAT;
    END.
```

Without an EXIT IF instruction, this program would repeat forever, until interrupted by ctrl-C, continually reading the position of the joystick and plotting a white point (the turtle) on the CRT.

24. MACHINE-LANGUAGE SUBROUTINES

We should also provide for machine-language subroutines. Two possible ways are: 1) use a special keyword, such as CALL, preceding a machine-language subroutine name, to indicate to the interpreter that this is a machine-language subroutine; and 2) begin each machine-language subroutine with a special keyword, and call it just like any other subroutine in the source program, i.e., by simply invoking it by name.

```
Example 1:
    ABC;
    CALL XYZ;
        :
    PROCEDURE ABC;
    (source-language subroutine)
    PROCEDURE XYZ;
    (machine-language subroutine)
Example 2:
    ABC;
    XYZ;
        :
    PROCEDURE ABC;
    (source-language subroutine)
    ML PROCEDURE XYZ;
    (machine-language subroutine)
```

Parameter passing to machine-language subroutines will present no problem when all procedures are computer generated, since they can follow standard conventions with the machine-language subroutines. Perhaps the best procedure is to pass a pointer (in a register, such as HL or IX) which points to a list of parameters. This table would contain, for each parameter, its type (REAL, INTEGER, COLOR, ARRAY of . . ., etc), whether it passes data *into* the procedure *only* or both *into* and *out of* the procedure, a pointer to where the variable is stored (or where the first element is stored, if an array or string), and, if it's a one-way parameter, where the procedure stores its local (changeable) copy of the variable.

If you don't think you need both one-way and two-way parameters, consider the following program segment:

```
    VAR A: INTEGER;
    PROCEDURE INCREMENT
        (I: INTEGER, VAR J: INTEGER);
    BEGIN;
            I: = I + 1;
            J: = I + 1;
    END:
    BEGIN;
            INCREMENT (6, A);
            PRINT (A);
    END;
```

In the above program segment, if parameter I is *not* one-way, execution of I: = I + 1; will cause the value of the constant 6 to be botched but if parameter J is not two-way, its value will never be inserted in variable A!!!

While the above example may seem to have a trivial solution in this particular case (just make the subroutine a function), the problem is *not* trivial for the general case where a subroutine may have to return MANY values! In fact, my programs usually contain several evaluative functions which return both the desired result *and* a BOOLEAN value which tells whether or not the required operation could be performed! Such a program segment usually looks like this:

```
    IF EVALUATE (A, B, C) THEN . . .
        ELSE WRITE ('ERROR. . .');
```

Such functions return the actual value as one of its parameters, with the function value itself being used as an error flag (most of the time). This form is especially useful when the evaluation is to be performed on user-typed input. In such a case, the IF is usually the EXIT of a LOOP which, if not exited, re-prompts and requests revised input. Remember, I firmly believe that NO possible user input should EVER cause a program to crash with an obscure error message!

CONCLUSION

From the preceding, you can see that I envision the creation of not just a TINY language, but a full-blown, general-purpose games and graphics language suitable for both tiny kids and adult game-writers. While younger kids will not be able to appreciate (or even understand) some of the more advanced features of the language, there is no reason they have to be taught the whole language at one! They can be started out with some of the simpler control structures and standard procedures, such as LOQP, IF, DRAW, and PRINT; and later on, introduced to the more advanced concepts.

Similarly, implementers with small systems (8K or so) may want to implement only the more basic features of the language, while those with larger systems may want to implement the whole language. Hopefully, some sort of standards could be set up specifying which parts to implement first, which parts second, etc., so that if someone advertises a program for Level 2 DRAGONSQUEAK, for instance, then ANYONE having Level 2 or higher, will be able to run the program with only minimal (mostly hardware-dependent) modifications, and not have to worry about structures, subroutines or features omitted from his Level 4 version (say) but present in someone else's Level 2 version.

Also, from the above examples, one can easily guess what my favorite language is! And while PASCAL may or may not be the ideal language on which to base a new language for kids (If anyone has a better idea, please speak up. . .that's what this whole thing is all about), it's certainly a good starting point. . . After all, when our kids outgrow DRAGONSQUEAK and are ready for something bigger and better (or are ready to go out into the real world of Incredible Big Monsters and Darned Expensive Computers), what would we rather they be using??? An anachronism like FORTRAN? An ungainly monster like COBOL? A kludge like BASIC??? Or a beautiful, natural, structured language like PASCAL!!! □

SPOT

The Society of PET Owners and Trainers

Photo courtesy of Visualscope.

PET SOCIETIES & NEWSLETTERS

Here's additional information about PET users' groups and newsletters. In the San Francisco Bay area, those interested in the East Bay's SPHINX society should contact Neil Bussey (415) 451-6364. Those in the San Jose-San Francisco area should call the Palo Alto Mr Calculator store at (415) 328-0740 for details on the next local users' group meeting.

Great news: the newsletter from the Bay Area groups is now available. It's packed with info that's available nowhere else. The most recent issue, for example, contained articles on a simple way to add a standard keyboard to the PET (while retaining PET graphics), an article on using the PET's 8-bit parallel I/O port, info on the PET's character set, memory map, and lots more, including announcements of many PET-related products. The two back issues are available at $.75 each; $4.50 will get the monthly newsletter for the next six months. Send orders to Pete Rowe, Lawrence Hall of Science, U.C. Berkeley, Berkeley, CA 94720.

THE PET PAPER is being published by Terry Laudereau, formerly Software coordinator for Commodore, and Rick Simpson, KIM Product Manager at MOS Technology, a Commodore company. It's scheduled to include articles to interest both beginners and experts, news of User Groups, software reviews, and hardware how-to's. For a year's subscription (number of issues not specified) send $15 to THE PET PAPER, PO Box 43, Audubon, PA 19407.

SOFTWARE

See our PET software review under 'Reviews'. Many distributors of PET software are springing up. Most offer royalty contracts for programs running from 2% of wholesale to 20% of retail. Many deal in both TRS-80 and PET programs. As of early April, these companies are marketing software:

Don Alan Enterprises, PO Box 401,
Marlton, NJ 08053.
Peninsula School Computer Project, Peninsula Way, Menlo Park, CA 94025.
Personal Software, PO Box 136-B4,
Cambridge, MA 02183; (617) 783-0694.
Silver State Enterprises, PO Box 27111,
Lakewood, CO 80227.
The PET Paper, PO Box 43, Audubon, PA 19407.

As of early April, these companies are gearing up to sell PET software:

Commodore, 901 California Ave, Palo Alto, CA; (415) 326-4000. Contact Adrian Byram.
Creative Computing, PO Box 789-M,
Morristown, NJ 07960; (201) 540-0445.
Kilobaud, Peterborough, NH; (603) 924-3873.
Mind's Eye Personal Software, PO Box 354,
Palo Alto, CA 94301; (415) 326-4039. (Run by Greg Yob, formerly of Commodore).

TEACHERS' (& KIDS') PET

We would like to communicate with other schools who are using the Commodore PET for educational purposes. We are a small K-10 school. We are currently teaching BASIC to some 7-10 graders and they are using the language to develop programs for their mathematics classes. We would be interested in sharing methods and programs with other schools who are attempting the same sort of thing.

Charles Ebert
The Midwestern Academy of
the New Church
73 Park Drive
Glenview, IL 60025

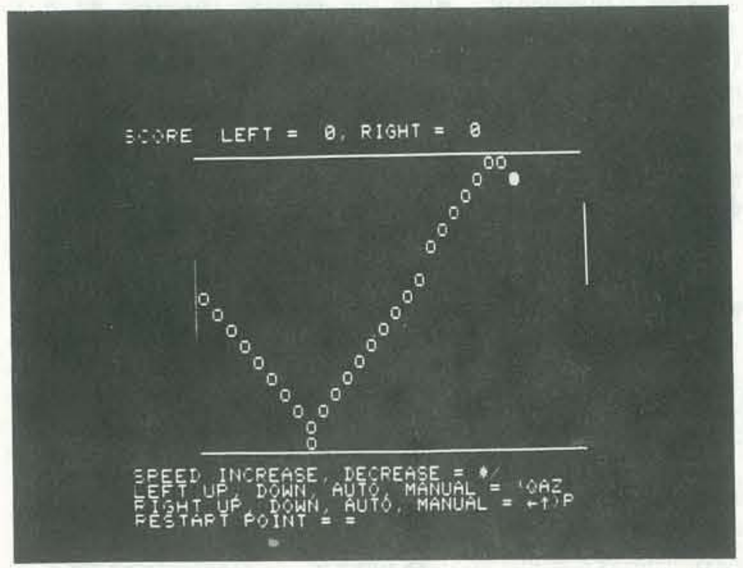
LISTING CONVENTIONS

Program listings employ the following conventions to represent characters that are difficult to print on a standard printer: Whenever square brackets appear in the listing, neither the brackets nor the text they enclose should be typed literally. Instead, the text between the brackets should be translated to keystrokes. For example, [CLR] means type the CLR key, [3 DOWN] means [DOWN, DOWN, DOWN] ie press the first CRSR key three times.

PONG for the PET

Martin Cohen, of Technology Service Corporation in Santa Monica, CA, has written a fine PONG game for the PET. The ball actually *squashes* when it hits a paddle or the 'floor' or 'ceiling' of the game room. You can increase or decrease the speed of the game to suit yourself. Since each paddle may be set either to automatic or manual mode you can vary the number of players from 0 to 2. Thanks, Martin!

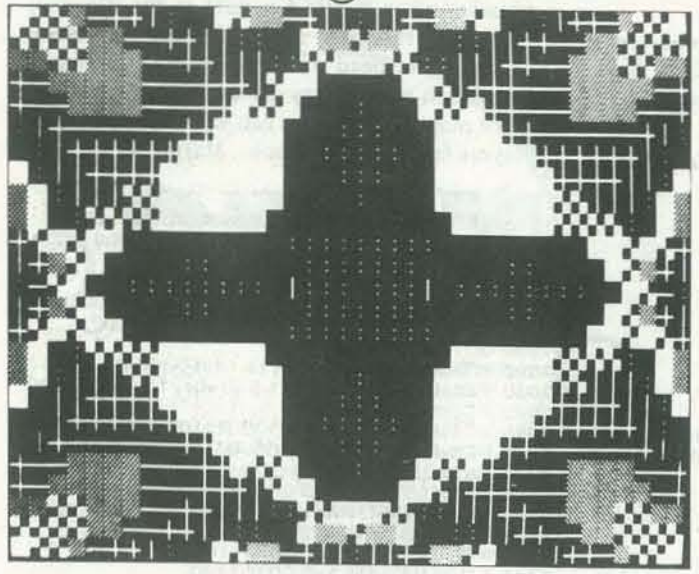
```
500 READ DS$,IS$,LU$,LD$,RU$,RD$,LA$,LM$,RA$,RM$,AG$
510 DATA "/","*","!","Q","←","↑","A","Z",")","P","="
515 Z3=32768:Z4=40
520 XO=5:X1=35
530 YO=2:Y1=19
540 LO=INT((YO+Y1)/2)-2:L1=LO+4
550 RO=LO:R1=L1:DM=0
560 PRINT"[CLR]";
570 Y2=Y1+1:Z2=99
580 FORX2=XOTOX1:GOSUB900:NEXTX2
590 Y2=YO-1:Z2=100
600 FORX2=XOTOX1:GOSUB900:NEXTX2
620 X2=X:Y2=Y
630 XP=X2:YP=Y2
640 GOSUB3000:GOSUB3100
650 LN=0:RN=0
660 DX=1.5:DY=0
670 PRINT"[HOME]";:FORI=1TOY1+2:PRINT"[DOWN]";:NEXTI
680 PRINT"SPEED INCREASE, DECREASE = ";IS$;DS$
690 PRINT"LEFT UP, DOWN, AUTO, MANUAL = ";LU$;LD$;LA$;LM$
700 PRINT"RIGHT UP, DOWN, AUTO. MANUAL = ";RU$;RD$;RA$;RM$
710 PRINT"RESTART POINT = ";AG$;
790 LA=0:RA=0
890 GOTO2400
900 REM PUT Z2 AT (X2,Y2)
910 POKEZ4*Y2+X2+Z3,Z2
920 RETURN
950 REM WAIT TJ SEC
960 TJ=TI+60*TJ
965 IFTI<TJGOTO965
970 RETURN
1000 REM TOP OF LOOP
1005 ZB=81:REM STANDARD BALL
1010 XX=X:YY=Y:X=X+DX:Y=Y+DY
1020 IFX<XOGOTO1200
1030 IFX>X1GOTO1300
1040 IFY<YOGOTO1400
1050 IFY>Y1GOTO1500
1060 REM TEST FOR KEY HIT
1070 GETC$:IFLEN(C$)>0GOTO2000
1080 IFLAANDDX<0GOTO1800
1090 IFRAANDDX>0GOTO1900
1100 REM DISPLAY AT(X,Y)
1120 XQ=XP:YQ=YP
1130 XP=INT(X):YP=INT(Y)
1140 ZQ=Z4*YP+XP+Z3
1170 IFZR<>ZQTHENPOKEZR,32:ZR=ZQ
1180 POKEZQ,ZB
1190 GOTO1000
1200 REM AT LEFT
1210 Y=Y-DY*(X-XO)/DX
1220 DX=-DX:X=XO:ZB=97
1230 IF(Y<YO)OR(Y>Y1)GOTO1040
1240 IF(Y<LO-.75)OR(Y>L1+.75)GOTO2200
1290 GOTO1600
1300 REM AT RIGHT
1310 Y=Y-DY*(X-X1)/DX
1320 DX=-DX:X=X1:ZB=225
1330 IF(Y<YO)OR(Y>Y1)GOTO1040
1340 IF(Y<RO-.75)OR(Y>R1+.75)GOTO2300
1390 GOTO1600
1400 REM AT TOP
1410 X=X-DX*(Y-YO)/DY
1420 DY=-DY:Y=YO:ZB=226
1430 GOTO1530
1500 REM AT BOTTOM
1510 X=X-DX*(Y-Y1)/DY
1520 DY=-DY:Y=Y1:ZB=98
1530 IFX<XOTHENX=XO
1540 IFX>X1THENX=X1
1550 GOTO1060
1600 REM MAKE BOUNCE FUNNY
1610 NB=NB+1:IFNB<5GOTO1040
1630 DD=SQR(DX*DX+DY*DY)
1640 DX=DX*(1.5*RND(1)+.5)
1644 AD=ABS(DY/DX)
1645 IFAD>2ORAD<.2THENDY=DX*(RND(1)+.5)
1650 DY=DY*((2-5/NB)*RND(1)+.7)
1660 DD=DD/SQR(DX*DX+DY*DY)
1670 DX=DX*DD
1680 DY=DY*DD
1690 GOTO1040
1800 REM AUTO LEFT
1810 IFY<LOTHENDM=-2:GOSUB3000:GOTO1100
1820 IFY>L1THENDM=2:GOSUB3000:GOTO1100
1830 GOTO1100
1900 REM AUTO RIGHT
1910 IFY<ROTHENDM=-2:GOSUB3100:GOTO1100
1920 IFY>R1THENDM=2:GOSUB3100:GOTO1100
1930 GOTO1100
2000 REM KEY HIT
2030 IFC$=DS$THENDX=DX/1.5:DY=DY/1.5:GOTO1100
2040 IFC$=IS$THENDX=DX*1.5:DY=DY*1.5:GOTO1100
2050 IFC$=RU$THENDM=-3:GOSUB3100:GOTO1100
2060 IFC$=RD$THENDM=3:GOSUB3100:GOTO1100
2070 IFC$=LU$THENDM=-3:GOSUB3000:GOTO1100
2080 IFC$=LD$THENDM=3:GOSUB3000:GOTO1100
2090 IFC$=RA$THENRA=1:GOTO1100
2095 IFC$=RM$THENRA=0:GOTO1100
2100 IFC$=LA$THENLA=1:GOTO1100
2105 IFC$=LM$THENLA=0:GOTO1100
2110 IFC$=AG$GOTO2400
2190 GOTO1100
2200 REM PASSED LEFT
2210 RN=RN+1
2220 GOTO2350
2300 REM PASSED RIGHT
2310 LN=LN+1
2350 REM SHOW WHERE SCORED
2360 X2=XP:Y2=YP:Z2=32:GOSUB900
2370 X2=INT(X):Y2=INT(Y):Z2=42:GOSUB900:TJ=TI+60
2380 IFTI<TJGOTO2380
2390 Z2=32:GOSUB900:GOTO2420
2400 REM A SCORE - DISPLAY AND START A POINT
2410 X2=XP:Y2=YP:Z2=32:GOSUB900
2420 PRINT"[HOME]SCORE: LEFT = "+STR$(LN)+",
     RIGHT = "+STR$(RN)+"[3 SPACE]";
2425 DD=SQR(DX*DX+DY*DY)
2440 R=RND(1):S=RND(1)+.5:DY=RND(1)
2450 IFR>.5THENX=XO:DX=S:Y=(LO+L1)/2
2460 IFR<=.5THENX=X1:DX=-S:Y=(RO+R1)/2
2470 XP=INT(X):YP=INT(Y):
2480 ZB=81
2490 DD=DD/SQR(DX*DX+DY*DY)
2500 DX=DX*DD:DY=DY*DD
2510 NB=0
2520 ZR=999
2540 TJ=1:GOSUB950
2550 X2=XP:Y2=YP:Z2=ZB:GOSUB900
2560 TJ=1:GOSUB950
2590 GOTO1100
3000 REM MOVE LEFT PADDLE DM
3010 X2=XO-1:ZP=103
3020 YA=LO:YB=L1:GOSUB3200
3030 LO=YA:L1=YB
3040 RETURN
3100 REM MOVE RIGHT PADDLE DM
3110 X2=X1+1:ZP=101
3120 YA=RO:YB=R1:GOSUB3200
3130 RO=YA:R1=YB
3140 RETURN
3200 REM MOVE A PADDLE
3210 Z2=32:Y8=Z4*YA+X2+Z3:Y9=Y8+Z4*(YB-YA)
3220 FORY2=Y8TOY9STEPZ4:POKEY2,Z2:NEXTY2
3230 YA=YA+DM:YB=YB+DM
3240 IFYA<YOTHENYB=YB+YO-YA:YA=YO
3250 IFYB>Y1THENYA=YA+Y1-YB:YB=Y1
3260 Z2=ZP:Y8=Z4*YA+X2+Z3:Y9=Y8+Z4*(YB-YA)
3270 FORY2=Y8TOY9STEPZ4:POKEY2,Z2:NEXTY2
3280 RETURN
9000 REM LINEARITY CHECK
9010 PRINT"[CLR, RVS]";
9020 FORI=1TO999
9030 PRINT"[shiftLBRACK]";
9040 NEXTI
9050 GOTO9050
```



```
SCORE  LEFT =  0, RIGHT =  0

SPEED INCREASE, DECREASE = */
LEFT UP, DOWN, AUTO, MANUAL = !QAZ
RIGHT UP, DOWN, AUTO, MANUAL = ←↑)P
RESTART POINT = =
```

To indicate motion in this photo we modified the program so the ball, shown as a white dot would leave a trail of 'hollow' dots. The 'trail' is not part of the program listed here.

KALEIDOSCOPE



```
4 ZC=0:C=0:ZQ=59456:ZW=32
5 PRINT "[CLR]";
6 CL(0)=ASC(" ")+128
7 CL(1)=ASC("[?]")-64
8 CL(7)=ASC(" ")
9 CL(3)=ASC("[@]")-128
10 CL(4)=ASC("[shiftLBRACK]")-128
11 CL(5)=ASC("[shiftRBRACK]")-128
12 CL(2)=ASC("[&]")-64·
13 CL(6)=ASC(":")
18 N1=32768: N2=40: N3=.625: N4=39.9999
20 FOR W=3 TO 50
30 FOR I=1 TO 19
40 FOR J=0 TO 19
50 K=I+J
60 C=CL((J*3/(I+3)+I*W/12) AND 7)
70 Y1=N1+N2*INT(N3*I)
80 Y2=N1+N2*INT(N3*K)
90 Y3=N1+N2*INT(N3*(N4-I))
100 Y4=N1+N2*INT(N3*(N4-K))
110 POKEI+Y2,C: POKEK+Y1,C: POKEN2-I+Y4,C
120 POKEN2-K+Y3,C: POKEK+Y3,C: POKEN2-I+Y2,C
130 POKEI+Y4,C: POKEN2-K+Y1,C
140 NEXT J
150 NEXT I
160 NEXT V.'
170 GOTO 20
```

Changes that prevent twinkling, but slow down display...

```
2 GOTO 4
3 WAITZQ,ZW:WAITZQ,ZW,ZW:POKEZC,C:RETURN
110 ZC=I+Y2:GOSUB3: ZC=K+Y1:GOSUB3: ZC=N2-I+Y4:GOSUB3
120 ZC=N2-K+Y3:GOSUB3: ZC=K+Y3:GOSUB3: ZC=N2-I+Y2:GOSUB3
130 ZC=I+Y4:GOSUB3: ZC=N2-K+Y1:GOSUB3
```

Kaleidoscope is a simple program that runs continuously while drawing interesting patterns on the screen of a Commodore PET computer. It is adapted from a program written by Rod Holt on a different computer.

You will probably want to try each of the two variations of the program. As originally written, you may see 'glitches' flashing on the screen while the program executes. You can get rid of these 'glitches' with a variation of this program provided by Larry Tesler. By replacing the POKEs with GOSUBs, as indicated, the program will slow down considerably, but the picture will be cleaner. I personally prefer the visual effects of motion that appear in the original, faster version.

You will notice that there are no PRINT statements in the program: instead, the program POKEs the ASCII equivalent of the graphic characters into the area of memory where the PET stores its current picture display. This area starts at memory location 32768. The first 40 locations of this area are for the first row of characters on the screen. The next 40 locations are for the next row, and so on.

The built-in function, ASC, does not quite give you the numbers you need for doing POKEs instead of PRINTs. If you are interested in experimenting with different graphics characters, the following statement, when executed, will tell you the integer that needs to be added or subtracted from the value ASC computes:

[CLR] ? –ASC("x")+PEEK(32775)

To assure that this statement will work, be sure not to include any spaces after you press the CLR key. Try different graphics characters in place of the x above. You are now prepared to change lines 6-13 with your own graphics characters.

If you want to change the *shapes* of the patterns created, replace

"J*3/(I+3)+I*W/12"

in line 60 with anything you please, and see what happens.

If you are interested in experimenting further, you can change PET's character set by executing POKE 59468,14. To restore the regular character set, POKE 59468,12. While the alternate character set is in effect, the characters generated by shift–) and shift– ← make for interesting kaleidoscope patterns.

Dave Offen
Menlo Park, CA

Tiny GRAPHICS

My children have enjoyed running the attached graphic programs on my PET. I am offering them in the hope others may enjoy them.

M C Hofheinz
Stockton, CA

```
A  10 POKE (32768 + 1000*RND(1)),
       255*RND(1)
   20 GOTO 10
       (Or substitute any number from 1 to
       255 for the expression after the comma.)

B  10 FOR X=1 to 255
   20 FOR Y=1 to 1000
   30 POKE (32, 767+Y), X
   40 NEXT Y
   50 NEXT X

C  20 FOR X=1 to 255
   30 PRINT X: POKE (32767+X), X
   40 NEXT X
       (Best to hold RVS during this one)

D  20 FOR Y=1 to 1000
   30 POKE (32767+Y); INT (Y/4+1)
   40 NEXT Y

E      Add to any of the above
   5   POKE 32768, 14
```

HAM PET OWNERS

Would you like to take part in experiments to transmit programs, etc by Ham Radio? Please get in touch with the undersigned. To arrange a schedule give frequency, time, date, and call letters and perhaps a telephone number.

Orin K Batesole–W6HJE
150 Shady Lane
Walnut Creek, CA 94596
Telephone (415) 934-8661

PET PRINTERS

A rumor of interest to all PET owners who have access to a Versatec printer: We hear that for $100 Versatec (Santa Clara, CA) will sell a Versatec printer/PET interface.

Commodore's $595 printer will allow you to print out graphic characters as they appear on the screen. It sure will be nice to be able to print graphics, but listings will still be confusing if a graphic character prints when cursor control is done in print strings. A sample of the print quality is shown below.

```
ABCDEFGHIJKLMNOPQRSTUVWX
ABCDEFGHIJKLMNOPQRSTUVWX
ABCDEFGHIJKLMNOPQRSTUVWX
```

DRAW UPDATE

1) A bug got into our last version: lines 7000 and 7040 should read

```
7000 PRINT "[CLR, DOWN]"
7040 PRINT "[HOME]";: NEW
```

2) A number of readers found line 5535 puzzling:

```
5535 : : V=C>BY: IF V=RV GOTO 5545
```

The leading colons are to force an indentation to make structure clearer. 'C>BY' is a Boolean condition. If C is greater than BY then the expression is TRUE, and so evaluates to –1: therefore V is set equal to –1. If C is not greater than BY, then the expression is false, and evaluates to 0: so V is set equal to 0. In the second command of the line, V is compared to RV in the usual manner.

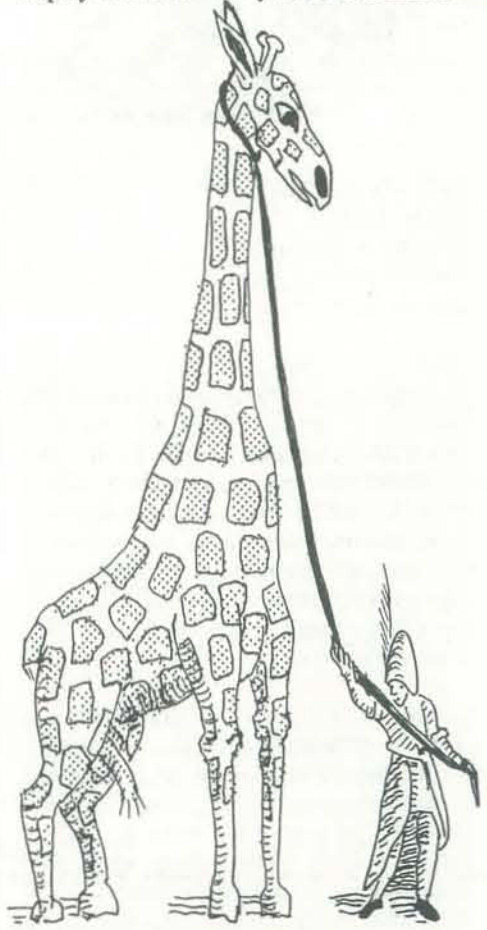
□

REVIEWS

PET SOFTWARE REVIEW

Don Alan Enterprises
P.O. Box 401, Marlton, NJ 08053
10 programs on a cassette, $19.95

Don Alan Enterprises is selling a PET cassette containing ten programs for $19.95. I am generally disappointed with the quality of these programs. However, since there is not yet available a wide selection of programs for the PET, there undoubtedly will be those of you who would rather play with these programs, than stare at the '7167 BYTES FREE' message displayed before you on the screen.



If you're a computer hobbyist who is just learning to program, and you are unfamiliar with the capabilities of small computers, you might appreciate this product. Most of the programs included are short. They provide readable examples of working programs written in the PET's particular dialect of BASIC.

Among the supplied programs are two requiring no intervention once they are started. One program transforms the computer into a digital clock with a large numerical display. The other, called WORM, draws a delightful criss-crossed maze of lines all over PET's display screen.

The remaining eight programs on the tape are interactive games. I find none of the games particularly inspiring. In addition, the authors do not devote nearly enough attention to the needs of the game-player. To me, this is a serious flaw because an important test of any good computer game is that it should be easy to interact with and pleasant to use.

In particular, the math practice program has no facility for letting you determine the difficulty level of the problems. How much value can there be in practicing on problems that may be either too easy or too difficult for you? Also, when you provide an invalid response to the initial question, the program prints out a confused and inappropriate message. This indicates a sloppy programming job.

I found shortcomings with some of the other programs as well. Some of the games unnecessarily require the player to press the return key after each single-letter response. Why should the programmer require that you press two keys when one is adequate? Other games would be considerably improved if they were to automatically repeat when a key is held down, rather than requiring twenty or thirty keystrokes on the same key.

The creators of this package of programs advertise that their product should be used to 'house-break your PET'. Unfortunately, you may discover that if you have an interest in moving beyond the toilet training stage, the Don Alan programs are not for you.

Reviewed by Dave Offen
Computer Software Consultant
Menlo Park, CA

STIMULATING SIMULATIONS

60 pp, $5.00

THE DEVIL'S DUNGEON

15 pp, $3.50

by C William Engel

Box 16612, Tampa, FL 33687

At school or at home, what do you do with your personal computer? Why, you write programs to make it do things, of course. But what things? One approach is to tackle problems directly related to school or work. You can learn a powerful lot of programming skills by developing software to multiply matrices or balancing end-of-month checking account statements. But this applications approach is less than edifying to the developing programmer who lacks meaningful applications suitable to his/her level of skill. An often overlooked alternative is the game-simulation. If the objective is learning to program, why not have fun doing it? There are lots of game-simulations available to be copied. *101 Computer Games*, edited by David Ahl, comes to mind. But this is a canned approach which emphasizes the recreational aspect of personal computing rather than skill development. C William Engel, in writing *Stimulating Simulations* has done a nice job of getting away from the copy-a-game approach. In this booklet, he offers ten game-simulations of varying difficulty. Judging from the accompanying scenarios, they are all exceedingly interesting. Dr Engel's contribution is to fully document each program with a scenario, a sample run, a very readable flow chart, a listing, and suggestions for minor and major changes. So what's new? Well, these programs are understandable. They can be decoded and modified by the learner-programmer. They can be rewritten for different systems or to do different things. In short, they teach!

The Devil's Dungeon is a more sophisticated game-simulation of the same genre as *Stimulating Simulations*. The game seems to be a variation of Caves. The objective of the game is to obtain a maximum amount of gold from the dungeon in the face of many hazards including monsters, and poisonous gas. It appears to have a high potential for interest and challenge. I can't say the *Devil's Dungeon* is in the same league as Star Trek, nor can I say that it isn't. What makes a computer game popular is often obscure. A reading of the scenario, however, and the high quality documentation more than warrants putting the *Devil's Dungeon* high on your things-to-try list.

Reviewed by Peter S Grimes
Curriculum Supervisor
San Jose Unified School District

Personal Software, (PO Box 136-B4, Cambridge MA 02183) offers Stimulating Simulations *on tape with Engel's book for $14.95. On one side of the tape are PET programs, on the other side TRS-80 programs.*

THE LITTLE BOOK OF BASIC STYLE

by John M Nevison

Addison-Wesley, 1978
147 pp, $5.95

There are numerous books out on programming style. Why should you read this one? Two reasons. One, this book is about style in BASIC programming. This is somewhat unique: most other books on style deal with more hospitable languages. Two, this book is specific. While most of the rules are generalities, the text is not. The author makes specific suggestions—indent this many spaces, put blank lines here—and so on.

I have one complaint—I don't like the author's programming style. A number of his suggestions do not agree with my (admittedly prejudiced) notions of style. However, you may not think so. As the author puts it, 'The person who cares enough about a program's style to argue with these rules probably has little need of them. On the other hand, an argument against a rule should be advanced for the same reason the rule itself was suggested: because there is a better way to make the program read.' I agree.

Reviewed by Eryk Vershen.

8080A/8085: ASSEMBLY LANGUAGE PROGRAMMING

by Lance A Leventhal

Osborne & Associates, Inc., 1978
400 pp, $7.50

This book comes as highly recommended as did Osborne and Associates' *An Introduction to Microcomputers, Volume 0: The Beginner's Book* (see Tom Williams' review in the March-April 1978 issue).

8080A/8085 is written in the same style as *Volume 0*, and it is everything I had hoped for in an instructional text on assembly language, as well as on how to use assembly language to program a microcomputer. It begins with a brief discussion of the meaning of instructions —the programming problem (program understandability, debuggability, entry speed, readability, and length), using octal versus hexidecimal, instruction code mnemonics, and advantages and disadvantages of high-level (as well as assembly) language. Next, there is a 'basic-literacy' discussion of assemblers and loaders, followed by thorough and concise definitions, descriptions, and examples of each instruction of the entire 8080A and 8085 instruction sets.

8080A/8085 goes one step further than *Volume 0* in that not only is it a primer, in the classical sense, full of examples and samples, but it is also an excellent reference manual, with sample macros, programs (one's complement, 8- and 16-bit addition/subtraction, word dis/assembly, sum of squares, and more), simple program loops, character-coded data, code conversion, arithmetic problems, tables and lists, subroutines, I/O devices and programs, interrupts—the list goes on and on. Chapters 14 and 15, on debugging, testing, documentation, and re-design, are, in themselves, worth the price of the book.

If you have (or plan to have) an 8080 microprocessor, and you want to program it in assembly language, *8080A/8085* is written especially for you. In short, the first twelve chapters concentrate on the writing of short programs; the rest describes how to formulate tasks as programs and how to put short programs together to form a *working* system.

Reviewed by Vicki Parish. □

Tiny BLACKJACK

BY MILAN CHEPKO

During a college computer programming course about 12 years ago, I wrote a very primitive Blackjack routine in PIL/L, a Basic-like language for a 360/50. Taking over 180 lines, it dealt the cards from a deck of 52, allowed the player to draw or stand, drew cards for the 'dealer', and then determined the winner. Over 10 years have passed, but I never forgot the hours of pleasure, sweating over a hot terminal while that magnificent beast sat in air-conditioned comfort down the hall!

Then about a year ago, I discovered that computers had shrunk both in size and price, and I started planning for one of my own (actually, it began as a digital clock for the office, but things got a little out of hand!). I settled on the 8080-A 'front panel' by Morrow's Micro-S, working into 8K of RAM with a VDM-1 and Morrow's cassette board handling the I/O. Incidentally, I was very impressed with the quality and the performance of George Morrow's boards—they go together easily, work reliably, and I have only begun to tap their capabilities.

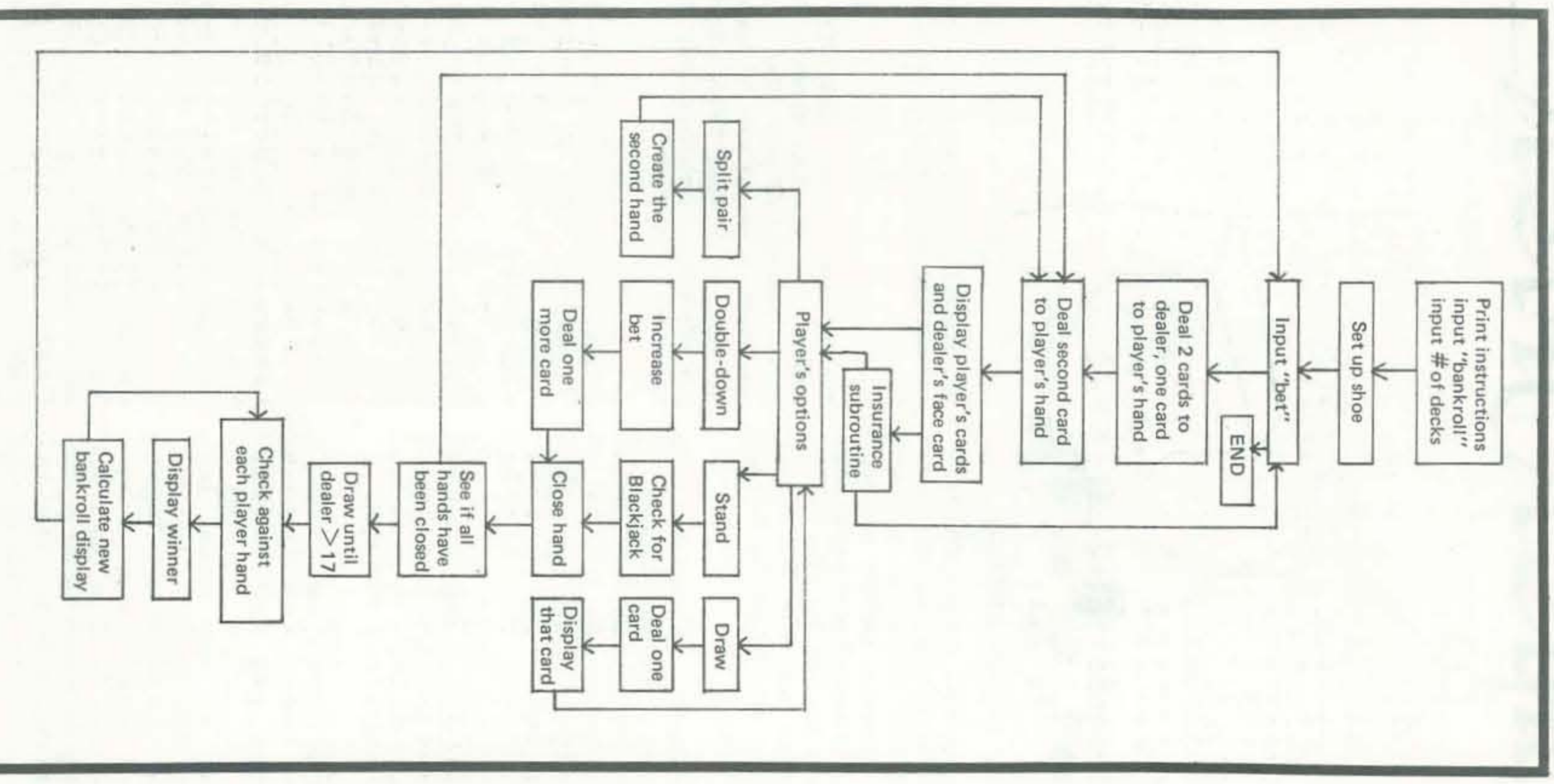
After 4 months of planning, building, and debugging hardware, I started playing with machine language and getting used to the 8080's instruction set by writing short subroutines. Eventually I came across Denver Tiny Basic by Fred Greeb (*Dr. Dobb's Journal*, March 76). The listing was in octal (essential, since I only had the octal pad provided by the front panel at the time), started at 000 000 (so no extensive re-write was needed), and included such features as a random number generator, multiple statements per line, and single-dimensioned variables. All this in less than 3K! Even with the VDM driver and some I/O routines, I still have over 4K left for programs in Tiny Basic.

All the Blackjack programs I've come across seem to require large amounts of memory, and generally leave out one or more functions that make the real game so interesting. This version allows splitting pairs and doubling-down, handles all betting, and even includes a small subroutine that lets the player see how many cards of each value remain in the shoe (equivalent to what players call 'casing the deck'). The listing totals 138 lines and just under 3400 bytes.

Most of the subroutines are self-explanatory, but there are a few features that could cause some confusion. First, I found that nothing is gained by displaying the suits (spade, heart, diamond, club) since they don't affect the point value of the cards. Therefore each deck contains 4 aces, 4 deuces. . . 4 kings. Lines 22–27 set up a 'shoe' containing the desired number of decks by establishing array S(), where each of the 13 elements contains an initial number of cards equal to 4 times the number of decks used. A card selection routine at line 160 then generates random numbers from 1 to 13, checks to see if any cards of that type remain in the shoe, subtracts one, and returns to the calling program.

The insurance routine (line 70) is activated when the dealer shows an ace at the beginning of play. This is an opportunity to protect your bet against the chance of the dealer having a Blackjack, although many players consider this to be a bad bet in general.

Standing, drawing, and doubling-down (doubling your initial bet in exchange for only drawing one card) are quite straightforward, but splitting pairs can get a little tricky. Basically (no pun intended!), you are turning one hand of 2 cards into two hands of one card each, then playing each hand separately from that point on. The program is written to allow 'nesting' hands 10 deep but I doubt you will ever have more than 3 or 4 hands in play. To simplify things, I arranged to play the highest-numbered hand to completion first, then the next-lower hand, until all hands are completed and it becomes the dealer's turn to draw. Since you can have another pair occur after splitting one pair, I had to use a flag to let the 'dealer' know when a hand was completed and prevent re-playing it. Therefore, at the end of each hand, 1000 is added to the total and stored for use later. The dealer knows that a hand is finished if the total exceeds 1000. This flag is subtracted to re-create the actual total for that hand.



Tiny Blackjack Structure

Blackjack is paid off differently than 21, and I needed a special flag to show when a Blackjack had occured. If all the conditions for Blackjack are met, line 131 converts the total to 100, then the 1000 is added as discussed above to close that hand. Later, after removing the 1000, a hand equal to 100 is identified as a Blackjack.

Casing the deck is an interesting subroutine. It doesn't really exist in casino play, unless you are blessed with a memory that can retain the cards as they are played. I included it for experimental purposes, but it could be left out without detracting from the game.

I noticed that the RND(0) function produces the same sequence of numbers when the game is started for the first time after being loaded into memory, so I included lines 14-16 to randomize this function. While any number up to the limit of Tiny Basic could be entered, large numbers produce excessive delays, and for practical purposes I use numbers up to 200 or 300. This could be likened to having the dealer open a new deck when you sit down at his table.

One note on debugging: to check out your version for typographical errors, I suggest you replace the random number generator at line 160 with IN X. . . this will allow you to set up hands of your choice, and then see how your program handles the situation. □

VARIABLES

A()	Players' cards
M()	Dealers' cards
B()	Bet on hand
P()	Players' total for each hand
C()	Cards of hand being played
S()	Shoe of cards
W	Wallet (how much money player has)
D	Number of decks in shoe
N	Number of cards left in shoe
I, J	Counters
H	Hand in play
F	Flag for insurance subroutine
O	Option
T	Total of points
X	Random number

RND(0) returns a number from 1 to 32767. CLRS clears the monitor screen. Could substitute PR:PR for use in a printer or teletype.

TINY BLACKJACK

```
           Dimensions Variables.   10  CLRS: DIM S(14), B(10), C(10), P(10), M(3), A(3)
                  Displays Rules.  11  PR" ***VEGAS BLACKJACK***"
                                   12  PR "DEALER STANDS ON ANY 17. . SPLITTING"
                                   13  PR "PAIRS AND DOUBLING DOWN PERMITTED"
                                   14  PR: PR"INPUT ANY NUMBER";
                                   15  IN R:I=0
                                   16  I=I+1: X=RND(0): IF I < R GOTO 16
                Inputs Bankroll.   20  PR "HOW MUCH CASH DO YOU HAVE";
       Sets up the shoe of cards.  21  IN W:PR "HOW MANY DECKS IN THE SHOE";
                                   23  IN D
                                   24  I=0
          When 20 or less cards    25  I=I+1: S(I)=4*D: IF I < 13 GOTO 25
         remain, resets the shoe.  26  N=52*D
                                   27  IF N < 20 PR "MUST SHUFFLE": GOTO 24
                                   30  I=0: F=0
                                   31  I=I+1: IF I > 10 GOTO 40
                                   32  B(I)=0: C(I)=0: P(I)=0: GOTO 31
           Inputs bet—if 0, goes   40  PR: PR "PLACE YOUR BET ($2 MINIMUM)";
                           to END  41  IN B(1): IF B(1)=0 GOTO 220
                                   42  IF B(1) < 2 GOTO 40
                                   43  W=W–B(1): H=1
        Draws 2 cards for dealer,  50  GOSUB 160
                  one for player.  51  M(1)=X: GOSUB 160
                                   53  M(2)=X: GOSUB 160
                                   54  A(1)=X
                  Draws 2nd card   56  GOSUB 160
                      for player.  57  A(2)=X
   Displays dealer's first card    58  CLRS:PR "HAND # ";H
      and both players' cards.     59  C(1)=M(1): PR "DEALER SHOWS ";
                                   61  I=1: GOSUB 151
                                   62  PR: PR "YOU HAVE THE FOLLOWING. ."
                                   63  C(1)=A(1): GOSUB 150
                                   64  I=2: C(2)=A(2): GOSUB 150
             Bypasses Insurance    65  IF H > 1  GOTO 80
                      Subroutine.  66  IF M(1) < 15 GOTO 80
                                   68  IF F > 0 GOTO 80
            Insurance Subroutine.  70  PR: PR "TYPE ANY DIGIT FOR INSURANCE, 0=NO;
                                   71  F=1: IN I: IF I > 0 W=W –B(1)/2
                                   73  IF M(2)=15 GOTO 80
                                   74  IF M(2) < 10 GOTO 80
                                   75  PR "DEALER HAS BLACKJACK!"
                                   76  IF 1 > 0 W=W+3*B(1)/2
                                   77  PR "YOU HAVE $"; W:GOTO 27
                Players' options.  80  I=2: PR: PR"HERE ARE YOUR CHOICES. ."
                                   81  PR" 0=STAND       2=DOUBLE DOWN"
                                   82  PR" 1=DRAW        3=SPLIT PAIR"
                                   83  PR"               4=CASE DECK"
                                   84  PR "CARD";
                                   85  IN O:IF O=0 GOTO 130
                                   87  IF O=1 GOTO 110
                                   88  IF O=4 IF C(3)=0 GOTO 230
                                   89  IF O=2 IF I=2 GOTO 120
                                   90  IF O=3 IF C(1)=C(2) IF I=2 GOTO 100
                                   91  PR "SORRY. .": GOTO 84
               Split Subroutine:   100  W=W–B(1):P(H)=C(1):C(1)=0
  Stores first card of the pair in 102  H=H+1: IF P(H) > 1000 GOTO 102
  P(H), moves second card into     103  B(H)=B(1):A(1)=C(2):C(2)=0
 A(1), then plays second hand.
                Draw subroutine:   104  GOTO 56
      Selects and displays a card. 110  GOSUB 160
  Calculates total to determine    111  I=I+1:C(I)=X:GOSUB 150
     "Bust" (could display the     112  GOSUB 140
              total if necessary). 113  If T > 21 PR"  ", "BUSTED!": GOTO 130
                                   114  GOTO 84
        Double-down subroutine.    120  W=W–B(1): B(H)=2*B(1): GOSUB 160
                                   121  I=I+1:C(I)=X: GOSUB 150
                                   130  GOSUB 140
              Stand subroutine:    131  IF T=21 IF I=2 IF H=1 IF P(2)=0 T=100
       Stores total + 1000 in P(H). 132  P(H)=T+1000: I=0
```

```
      Checks to see that all hands 133  I=I+1: C(I)=0: IF I < 10 GOTO 133
                   are completed.  134  H=H–1: IF H=0 GOTO 170
                                   135  IF P(H) > 1000 GOTO 134
                                   136  A(1)=P(H): GOTO 56
                                   140  J=0: T=0
            Addition subroutine:   141  J=J+1: IF C(J)=0 GOTO 145
       Adds point value for each   142  IF C(J)=15 T=T+11: GOTO 141
           card. . .if exceeds 21, 143  IF C(J) < 11 T=T+C(J): GOTO 141
        checks to see if any Aces  144  IF C(J) < 14 T=T+10: GOTO 141
   can be reduced from 11 to 1.    145  J=0: IF T < 22 RET
                                   147  J=J+1: IF C(J)=0 RET
                                   148  IF C(J)=15 C(J)=1: GOTO 140
                                   149  GOTO 147
                Print subroutine.  150  PR " ", " ";
                                   151  IF C(I)=15 PR"ACE(1 OR 11)": RET
                                   152  IF C(I) < 11 PR C(I): RET
                                   153  IF C(I)=11 PR "JACK": RET
                                   154  IF C(I)=12 PR "QUEEN": RET
                                   155  IF C(I)=13 PR"KING": RET
     Generates random numbers.     160  X=RND(0)/1000+1
      Checks to see if any cards   161  IF X > 13 GOTO 160
  of that type are left in shoe.   162  IF S(X)=0 GOTO 160
                                   163  S(X)=S(X) –1:N=N –1
                                   164  IF X=1 X=15
                                   165  RET
              Dealer subroutine:   170  CLRS: PR "DEALER HAS THE FOLLOWING. ."
        Draws cards for dealer     171  C(1)=M(1): C(2)=M(2)
         until total exceeds 16.   172  I=I: GOSUB 150
                                   173  I=2: GOSUB 150
                                   174  GOSUB 140
                                   175  PR "TOTAL= "; T
                                   176  IF T=21 IF C(3)=0 T=100: GOTO 190
                                   177  IF T >16 GOTO 190
                                   178  GOSUB 160
                                   179  I=I+1: C(I)=X: GOSUB 150
                                   180  GOTO 174
            Win-Lose subroutine:   190  M(1)=T; H=0
              Determines winner    192  H=H+1
                   for each hand.  193  P(H)=P(H) –1000: IF P(H) < 0 GOTO 27
                                   194  PR: PR "HAND #"; H
                                   195  IF M(1)=100 PR"DEALER HASBLACKJACK":GOTO 197
                                   196  PR "DEALER HAS "; M(1)
                                   197  IF P(H)=100 PR "YOU HAVE BLACKJACK":GOTO 200
                                   198  PR "YOU HAVE "; P(H)
                                   200  IF P(H)=100 IF M(1)=100 W=W+B(1):GOTO 210
                                   201  IF P(H)=100 W=W+5*B(1)/2:GOTO 211
                                   202  IF M(1)=100 GOTO 212
                                   203  IF P(H) > 21 GOTO 212
                                   204  IF M(1) > 21 W=W+2*B(H):GOTO 211
                                   205  IF P(H)=M(1) W=W+B(H): GOTO 210
                                   206  IF P(H) > M(1) W=W+2*B(H) : GOTO 211
                                   207  IF P(H) < M(1) GOTO 212
                                   210  PR "NO WINNER. . . YOU HAVE $ "; W:GOTO 192
                                   211  PR "YOU WIN!!. . . YOU HAVE $ "; W:GOTO 192
                                   212  PR "YOU LOSE!. . . YOU HAVE $ "; W:GOTO 192
                              END  220  CLRS: PR "YOU HAVE $ "; W:IF W > =0 END
                                   222  PR "THE MAFIA WILL PROBABLY ARRANGE"
                                   223  PR "AN 'ACCIDENT' FOR YOU!!!":END
                       Case deck:  230  CLRS: PR "THESE CARDS REMAIN. .":PR
         displays number of each   232  PR "2'S", S(2)
   type of caid left in shoe—      233  PR "3'S", S(3)
     waits for several seconds     234  PR"4'S", S(4), " ", "ACES", S(1)
             and then returns to   235  PR "5'S", S(5)
                 calling program.  236  PR "6'S", S(6), " ", "10'S",S(10)+S(11)+S(12)+S(13)
                                   237  PR "7'S",S(7)
                                   238  PR "8'S",S(8)
                                   239  PR "9'S", S(9)
                                   240  J=0
                                   241  J=J+1: IF J < 70 GOTO 241
                                   242  GOTO 58
```

IN DEFENSE of the COMPUTER ESTABLISHMENT

BY WARNER MACH

We're pleased that reader Mach has taken up the challenges raised by Jacques Vallee in a recent article. Mr Mach is Systems Analyst/Technical Manager at the Detroit Board of Education, and currently finishing requirements for a Master's Degree in Computer Science at Wayne State University.

I read Mr Vallee's article (Nov-Dec 1977 issue of *People's Computers*), 'There Ain't No User Science', which was billed as a 'tongue-in-cheek' discussion of difficulties on computer nets caused by programmers and other computer types. The discussion seemed less 'tongue-in-cheek' than a straightforward list of complaints.

Since I am a (gasp) Systems Programmer on a (booo) IBM machine and have worked a number of years in the educational environment I would like to defend the BAD GUYS. I would also like to confess that I am also a longtime (BA – Before Altair) subscriber to *People's Computer Co./People's Computers* and have my own KIM (so I am not totally mindlessly dedicated to the Intimidating Bad Machine).

I would like to rebut some of the specific notions in the article, but even more I would like to expose a sort of curious attitude on the part of certain elements of the hobbyist/educational fraternity concerning the motives of the establishment computer people.

MAINTAINING THE POWER

This curious attitude is well expressed by Mr Vallee and by Ted Nelson (*Computer Lib / Dream Machines*). The general notion is based on the following presumptions:

1. Computers are basically simple.
2. There is a group of people who are deliberately making it difficult for the Poor Suffering User (hereafter known as PSU) to use the computer.

This is being done because:

A. The Establishment Priesthood (hereafter known as EP) wants the ego gratification of forcing the users to come to them for answers.
B. The EP enjoys the power and control which comes of being the only ones who know what is going on.
C. The EP is afraid they will lose their jobs if the masses learn to fare for themselves.
D. For some reason the EP attracts a particularly noxious type of person who enjoys forcing PSU's to perform unnatural and inhuman tasks.

Control is maintained by:

A. Inventing secret languages full of 'Computercrud' (Nelson) and 'Obfuscation' (Vallee).
B. Creating artificial barriers to easy machine access.
C. Imposing ill-fitting systems.
D. Being non-responsive and obstinate when facing user requests.

FINDING THE VILLAINS

In looking at these charges, we first have to determine who comprises the PSU's and who is the EP. If I am the Systems Programmer on an IBM machine then am I really part of the EP because I delight in torturing the students and teachers who are my PSU's? Or am I really a PSU myself since I am under the Ultimate EP: IBM? How much secret lore do I have to ingest before I cross the border between PSU and EP? And how about Mr Vallee . . . does he not sometimes find himself in the role of EP as he explains, for example, how to put paper in a terminal?

Let's assume for a minute that, in fact, in the course of a computer-associated career that a person will likely find himself at various times on one side or the other of the fence. Let's go even one step further and pretend, for the sake of argument, that computers work pretty much like everything else in our experience; other pieces of machinery like, for example, cars.

FACING REALITY

1. Reliability is a function of experience. In the early days of cars if you wanted to go any distance you anticipated lots of flat tires and breakdowns (sort of like system crashes). As more experience was gained, cars became more reliable.

2. Economics determines what is possible. It is particularly astonishing to me that much of the villainy ascribed to the EP is simply a matter of economics. In addition to the direct economic aspect (how many programmers are we willing to hire and what kind of resources are we willing to devote), economics appears, directly or subtly, in almost anything that does or does not get done on a computer.

   The other day I saw a PLATO terminal for the first time . . . an incredible terminal with incredible software support. Of what use is it for me to compare that $6000-$10000 terminal tied to a $1000-a-month network with my ADM-3A tied to a $100-a-month network?

   For some reason the same people who buy a Ford and don't expect it to act like a Fiat expect that all software should be able to do anything . . . perhaps this is because (a) products of thought are somehow 'less real' than manufactured items and (b) it is 'theoretically' possible for any software to emulate any other.

3. Programs are made by people.
   If you have to 'list' your file when you are not under the editor and you have to 'print' your file when you are under the editor there are two possible ways this might have come about:
   A. Conspiracy theory:
      'OK folks, how can we confuse the user and maintain our position in the EP . . .'
   B. Project management not as tight as it should be:
      Joe Epsidic of the Editor Team talks to his superior: 'Hey Pete. . . What command should I use to type out the file?' 'I don't give a damn. . . Use "print". . . When you gonna finish that routine?'

      Larry Ascii, of the File-Control Team, is simultaneously talking to the programmer across the desk . . . 'What you think we should say to type out the statements?' 'How about "list" . . . it's easy to remember.'

4. Humans are bad prophets and have access to limited information.
   The IBM 360/370 operating systems, for example, were *very large* software projects. In order to accomplish the task, each programmer (as in any large programming task) was given a small portion of the code to work on, along with information as to the parameters which would be passed to him and the parameters which he should pass out of his program. A programmer working in such an environment codes things like error messages in such a way as to make them meaningful within his portion of a larger project. Not having a broad overview of the system as a whole, he has no way of predicting exactly how his coded message will appear to the end user, and indeed no precise idea what it will ultimately come to mean! Under these circumstances the best the systems programmer can do is to document in detail the conditions that may cause the message to appear while avoiding oversimplifications that may well be misleading.

ANTICIPATING USER FRUSTRATIONS

But enough of defending the coders of operating systems. Let's move on to how to 'anticipate' user frustrations.

According to the article: 'Never start implementing a system until the end users have been identified and given easy access to the designers . . .' This is a sort of motherhood-and-apple-pie statement, but what does it mean? The implication is that the EP is in the habit of arbitrarily designing (or mis-designing) systems which it then forces down the throats of the PSU's. As anyone who has designed a system knows, one of the very most difficult things to determine is what the end-user needs. The reason this is difficult to determine is *not* (generally) because the EP prefers to misdirect its energies as opposed to meeting the needs of the PSU, but rather because the user simply doesn't know what he needs and what the computer can and cannot do for him.

'Aha!' I hear someone exclaim. 'Spoken as a true patronizing member of the EP.' But it's true, and there is a large amount of literature devoted to the slippery problem of how to achieve a reasonable interface between the user and the computer. It is fair to say that, far from resenting the intrusion of the user, a systems analyst of any competence would probably bathe in oil (warm) the feet of a PSU who would come to him with an accurate documentation of the system in a form which could be readily implemented on the computer (said user presumably having ironed out all political problems which, often as not, are the biggest difficulty).

In Vallee's article he was talking about a computer net. If this net is to be available to anyone with the money and inclina tion to sign up for the service (as opposed to a net initially financed by a specific group or groups for a specific purpose) then how are the end users to be identified in advance of the several-year implementation effort? . . . Once the service is available then the clients will appear. To ask them to appear in advance is somewhat trickier than trying to talk to the drivers who will be using a proposed freeway. It almost sounds as though Mr Vallee bought into a net after it was already in operation and was irritated because he wasn't consulted in its design!

Another user frustration indicated is excessive non-comprehensible typing which is required. I am inclined to agree that a user should only type what is necessary (does anyone disagree?). The interactive systems I am familiar with (VM, MTS, TECHNOTEC) require the user identification and password, which is a minimum. I think there are a lot of systems like this.

I somewhat disagree with the notion of '. . . never give him (the PSU) an output that is outside the task context . . .'. I disagree because in many instances a precise explanation of the problem is required for a solution, and a more precise statement for the sake of the EP may be less understandable to the PSU. The question is whether the more precise statement is eventually to the PSU's benefit.

Generally a conversation with a PSU runs something like this (on the phone):

PSU: It doesn't work.
EP: What doesn't work?
PSU: The computer.
EP: What are you running?
PSU: Not running anything . . . It doesn't work.
EP: I mean, were you trying to run BASIC or send a job to the batch machine, or what?
PSU: I just dialed this number glued on the terminal and it doesn't work . . .
EP: Did you hear a high-pitched tone when you dialed?
Etc.

Believe me, even though an output may mean nothing to the user it very frequently means a whole lot to the EP representative who, hopefully, is trying to help (it may very likely be the only scrap of concrete information around). There may be, perhaps, other ways of getting this information to the EP than have the PSU convey it verbally from his terminal printout, but this is, by far, the easiest and the quickest. I wonder, too, if a more precise explanation of a problem may be irritating to the PSU initially but might be appreciated as he gains more experience with the system.

Another issue raised by Mr Vallee is the so-called 'wide angle fallacy'. I find this notion rather at variance with the other things he has said. Evidently his group arbitrarily and non-democratically decided to restrict the commands available to the PSU's for-their-own-good (I doubt if they consulted with the PSU's about this . . . The usual inclination of PSU's is to ask for everything they ever heard of). Apparently, a determination was made of the most frequently used commands and only those pages of the manual were passed out to the PSU's . . . He seems to regard this as a major accomplishment. Except for, presumably, a little disk space did it hurt that the additional, unused commands were available? Is it possible that more experienced users of the net did use the additional commands?

Users generally pass through three stages:

1. *Need help stage:* At this stage many prompting and 'help' facilities should be available to the user. Commands should be few and simple.
2. *Experienced stage:* At this stage the prompting should be infrequent. The user should be provided with abbreviated commands and shortcuts. Specialized commands can be introduced.
3. *Super whiz:* User is familiar with whole battery of specialized commands. Uses abbreviations for all common commands. Perhaps provided with an 'extensible' facility that allows him to tailor his own commands.

The stage reached by a user is determined by the amount of experience in terms of the number of hours logged and frequency of use. Professional users of the net (who most likely would be catered to – economics again) would be dissatisfied with a restricted subset of commands.

THE CASE OF THE INDIFFERENT EP MANAGER

I was amused by the dialog between a PSU who wanted to change the message given to the user during an interrupted session and the manager of the network facility. Mr Vallee presents this as though the EP manager, in the perverse manner of EP people everywhere, saw it as his duty to mold the PSU into an unnatural shape. Since I have been on the opposite side of the table from a PSU from time-to-time, I know what was going through the head of the manager:

1. There are X (units, tens, hundreds) of PSU's out there in user land, all of whom have at least one idea of how the system should be changed. If the floodgates were opened, with our present staff we would be programming and documenting to the year 3000.
2. Any programming change, no matter how small, endangers the whole net. Is it worth endangering the net for this request? (Remember from Mr Vallee's survey that system crashes are the thing that disturbs PSU's the most. . .) It doesn't take long for a programmer to develop the general philosophy of 'If it works don't change it'.
3. It is difficult to predict how long it will take to make a programming change (even a simple one). There will be the expense (economics again) of the programmers' salaries, plus documentation costs, plus documentation distribution costs.
4. This change may be important to this user but how 'visible' is it? (It may be better to ask for major enhancements to the system than minor improvements that can't be seen to sell anything. . .) Maybe other users will be unhappy with the change.

I've got to say that the manager's PR technique needs improvement. My technique is to pull out my 'list of things that need doing'. . . I then say, 'That's a good idea, but I don't know how soon we'll get to it' as I add the new entry to the bottom of the list. (This is a real list, by the way. It is conceivable, though unlikely, that all entries will eventually be processed).

Has there ever been a PSU who said, 'We think that this change is so important that we will pay any costs associated with implementing it and we will not complain if the system crashes as a consequence of trying to put it in?'

THE TALE (OR TAIL) OF THE CRASH

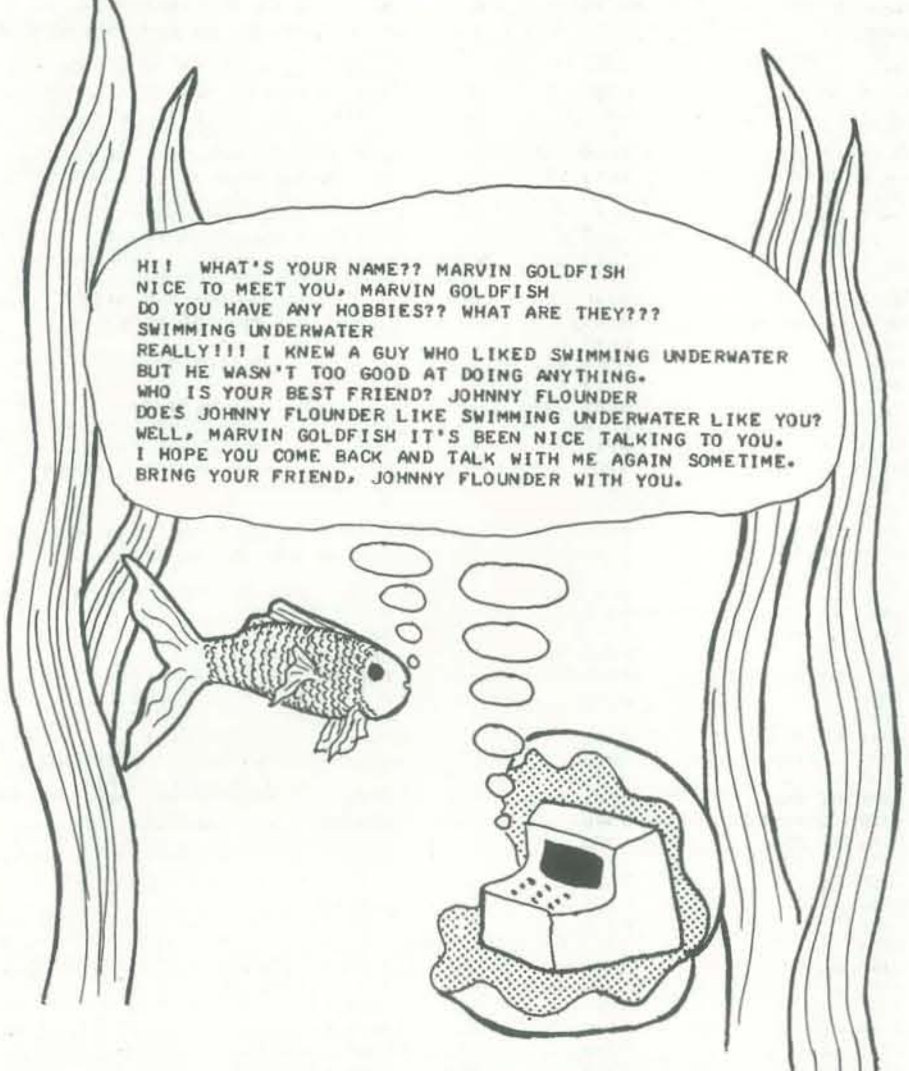
Another feature of the article which was sort of amusing was the account of the system crash. At first I was a little puzzled why the discussion of what trans-

Continued on page 53.

BASIC5 strings

BY FR. THOMAS MCGAHEE

Many SOL 20 owners have suffered along without string capabilities while waiting delivery of Processor Tech's 8K BASIC. But Father McGahee found time to write a string handler for BASIC5, so as to give his students capabilities for conversational-type programs such as the one illustrated on this page.



```
HI!  WHAT'S YOUR NAME?? MARVIN GOLDFISH
NICE TO MEET YOU, MARVIN GOLDFISH
DO YOU HAVE ANY HOBBIES?? WHAT ARE THEY???
SWIMMING UNDERWATER
REALLY!!! I KNEW A GUY WHO LIKED SWIMMING UNDERWATER
BUT HE WASN'T TOO GOOD AT DOING ANYTHING.
WHO IS YOUR BEST FRIEND? JOHNNY FLOUNDER
DOES JOHNNY FLOUNDER LIKE SWIMMING UNDERWATER LIKE YOU?
WELL, MARVIN GOLDFISH IT'S BEEN NICE TALKING TO YOU.
I HOPE YOU COME BACK AND TALK WITH ME AGAIN SOMETIME.
BRING YOUR FRIEND, JOHNNY FLOUNDER WITH YOU.
```

```
10    GOSUB 30000: REM * CLEAR STRING STORAGE AREA
20    PRINT "HI!  WHAT'S YOUR NAME?? ";: GOSUB 10000: N=Z
25    PRINT
30    PRINT "NICE TO MEET YOU, ";: Z=N: GOSUB 20000
35    PRINT
40    PRINT "DO YOU HAVE ANY HOBBIES?? WHAT ARE THEY???"
50    GOSUB 10000: H=Z
55    PRINT
60    PRINT "REALLY!!! I KNEW A GUY WHO LIKED ";: Z=H: GOSUB 20000
65    PRINT
70    PRINT "BUT HE WASN'T TOO GOOD AT DOING ANYTHING."
80    PRINT "WHO IS YOUR BEST FRIEND? ";: GOSUB 10000: F=Z
85    PRINT
90    PRINT "DOES ";: Z=F: GOSUB 20000: PRINT " LIKE ";
95    Z=H: GOSUB 20000: PRINT " LIKE YOU?"
100   PRINT "WELL, ";: Z=N: GOSUB 20000
110   PRINT " IT'S BEEN NICE TALKING TO YOU."
120   PRINT "I HOPE YOU COME BACK AND TALK WITH ME AGAIN SOMETIME."
130   PRINT "BRING YOUR FRIEND, ";: Z=F: GOSUB 20000: PRINT " WITH YOU."
140   PRINT : END
10000 Z=CALL(16384): RETURN
20000 Z=ARG(Z): Z=CALL(16442): RETURN
30000 Z=CALL(16430): RETURN
```

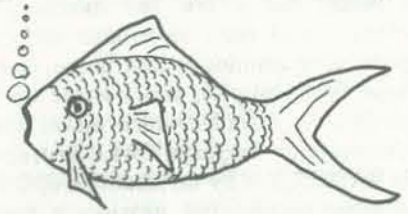
Our school recently purchased a SOL 20 from Processor Tech. I assembled it, and we are now using it in our computer course here at Don Bosco Tech. We have the 8K BASIC on order, but while we are waiting for that we have been happily programming away using BASIC5. One of the things that BASIC5 is missing is strings. Too bad, 'cause strings are lots of fun to use in programs to provide a more conversational feedback and 'personal' sounding program.

I finally had a few free moments the other day (I teach electronics and computer programming at Don Bosco, and am kept fairly busy!!), and I wrote up this short string-handler which makes use of the machine language CALL instruction in BASIC5. It is by no means an optimum implementation, but provides a reasonable flexibility. I will be doing up a more useful version soon, but in the meantime I figured maybe the guys and gals at PCC might be interested in this first version. I guess there are a lot of SOLs out there with BASIC5, and not all of the users are capable of doing up their own string handlers. . . so they might like to try this one out until something better comes along.

I assembled my particular version starting at 4000 hex (16384 decimal). The assembler used was the ALS-8 from Processor Tech. I tried to keep things simple. To input an ASCII string the user does a CALL to ASCIN. This routine starts storage at the next available location in the text storage area, which is pointed to by LAST. It duplicates this address in BEG (for BEGINNING) for later use in setting the BC registers prior to a return to BASIC. I use the SOLOS input routine at 0C01F to get keyboard input, then I strip off the MSB (parity bit) since otherwise TTYs might give us codes different from some keyboards. The ASCII is then stored in memory and the current address updated to point to the next available location. At this time (before any echoing), a check is done to see if the

ASCII character was a Line Feed (LF). I use the line feed as a terminator rather than Cairiage Return (CR), because this allows the user to input extremely long strings, such as entire poems and the like!! If it was not a LF then the character is placed in the B register and echoed using the SOLOS routine at 0C019. Since the echo causes the A register to be changed, but B still has the ASCII code, we copy B into A so we can perform comparisons. A CR will result in a CR, LF, and one NULL being sent out. If the user has made a mistake, he may type in a DELETE, which will cause the program to back up the memory to the proper place. Input continues uninterrupted until a Line Feed is finally typed.

When input is done, the present address (next empty location) is stored in LAST so the next time ASCIN is used it will start off at the right place. The ORIGINAL BEGINNING of the present text string is



then recovered from BEG and transferred to the B and C registers, since the BASIC CALL instruction uses these registers for transferring data between BASIC5 and the machine language routines. Then there is a RETurn to BASIC5. You will notice that there is a special entry point labeled INIT. Upon entry here the DONE portion of ASCIN is used to reset the address pointers to the beginning of the text storage area. This entry point can be used at the beginning of a BASIC program to 'clear' the string storage area. (Notice that it does not erase anything. . . it merely allows us to recycle storage space to conserve memory.)

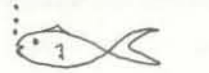
The ASCII output routine operates by taking the address found in the B and C registers and setting that up as the current address for memory. (The B and C registers are loaded with the address prior to the BASIC CALL using the ARG instruction. . . see sample program for details). The program now starts extracting ASCII characters one at a time and printing them. A CR will again result in a CR, LF, and NULL, using the same subroutine used during input. When a Line Feed is finally encountered, there is a RETurn to BASIC5. The Line Feed is NOT printed.

```
4000                 0010 * MACHINE LANGUAGE ROUTINES TO ADD STRINGS
4000                 0020 * TO BASIC5 VIA "CALL" INSTRUCTIONS.
4000                 0025 * WRITTEN BY FR. THOMAS MCGAHEE
4000                 0030 * ELECTRONICS AND COMPUTER INSTRUCTOR
4000                 0035 * DON BOSCO TECH, PATERSON, NEW JERSEY 07502
4000                 0040 *
4000                 0100 *** ASCII INPUT WITH ECHO.
4000 2A 5A 40        0105 ASCIN  LHLD   LAST    RECOVER ADDRESS
4003 22 5C 40        0110        SHLD   BEG     STORE FOR LATER USE
4006 CD 1F C0        0115 INP    CALL   0C01FH   GET A CHARACTER
4009 CA 06 40        0120        JZ     INP     CHECK STATUS
400C E6 7F           0122        ANI    7FH     MASK PARITY BIT
400E 77              0125        MOV    M,A     STORE IN MEMORY
400F 23              0126        INX    H       UPDATE CURRENT ADDRESS
4010 FE 0A           0127        CPI    0AH     IF A LINE FEED...
4012 CA 31 40        0128        JZ     DONE    ...PREPARE TO RETURN
4015 47              0130        MOV    B,A     PUT IT IN B FOR SOLOS...
4016 CD 19 C0        0135        CALL   0C019H   ...SO IT CAN ECHO IT
4019 78              0140        MOV    A,B     IN "A" FOR COMPARES
401A FE 0D           0150        CPI    0DH     IF A CARRIAGE RETURN...
401C CC 4E 40        0155        CZ     CR      ...THEN DO LF AND NULL
401F FE 7F           0170        CPI    7FH     "DELETE" NEEDS HELP
4021 C2 06 40        0175        JNZ    INP     BACK FOR MORE!
4024 06 01           0185        MVI    B,01H   ...B HAS BACKSPACE...
4026 CD 19 C0        0190        CALL   0C019H   ...PRINT A BACKSPACE...
4029 2B              0192        DCX    H       DOUBLE DECREMENT...
402A 2B              0193        DCX    H       ...CLEARS BAD DATA
402B C3 06 40        0195        JMP    INP     ...AND GET MORE!
402E                 0197 *
402E 21 63 40        0200 INIT   LXI    H,TXT   *RESET POINTERS
4031                 0203 *
4031 22 5A 40        0205 DONE   SHLD   LAST SAVE FOR NEXT TIME
4034 2A 5C 40        0210        LHLD   BEG     GET "ORIGINAL" ADDRESS..
4037 44              0215        MOV    B,H     ...AND STORE IN B,C
4038 4D              0220        MOV    C,L     ...FOR BASIC5 LINKAGE
4039 C9              0225        RET    BYE-BYE!
403A                 0227 *
403A                 0230 ***ROUTINE TO OUTPUT STORED ASCII STRINGS
403A 60              0235 ASCIO  MOV    H,B     TRANSFER ADDRESS ...
403B 69              0240        MOV    L,C     ...IN B,C TO H,L
403C 7E              0245 OUT    MOV    A,M     GET STORED CHARACTER
403D 47              0250        MOV    B,A     STORE IN B FOR NOW
403E FE 0A           0255        CPI    0AH     LF NOT PRINTED
4040 C8              0260        RZ     LF MEANS GO HOME!
4041 CD 19 C0        0265        CALL   0C019H   PRINT CHARACTER
4044 23              0270        INX    H       SET NEW ADDRESS
4045 78              0275        MOV    A,B     NEED IT IN "A"
4046 FE 0D           0280        CPI    0DH     CR NEEDS HELP
4048 CC 4E 40        0285        CZ     CR      SO HANDLE IT WITH CARE
404B C3 3C 40        0290        JMP    OUT     GO FOR MORE OUTPUT
404E 06 0A           0295 CR     MVI    B,0AH   WITH A CR YOU GET...
4050 CD 19 C0        0300        CALL   0C019H   ...A FREE LINE FEED...
4053 06 00           0305        MVI    B,00H   ...AND A FREE NULL...
4055 CD 19 C0        0310        CALL   0C019H   ...TO ALLOW CLEAN I/O
4058 78              0320        MOV    A,B     NO TRASH, PLEASE
4059 C9              0325        RET    THAT'S ALL, FOLKS!
405A                 0326 *
405A                 0327 * STORAGE AREA FOLLOWS
405A 63 40           0330 LAST   DW     TXT     STORAGE
405C 63 40           0335 BEG    DW     TXT     STORAGE
405E 00              0340        NOP    FREE LOCATION
405F 00              0345        NOP    FREE LOCATION
4060 00              0350        NOP    FREE LOCATION
4061 00              0355        NOP    FREE LOCATION
4062 00              0360        NOP    FREE LOCATION
4063 00              0365 TXT    DB     00H     TEXT STORAGE BEGINS

ASCIN    4000
ASCIO    403A
BEG      405C      0110 0210
CR       404E      0155 0285
DONE     4031      0128
INIT     402E
INP      4006      0120 0175 0195
LAST     405A      0105 0205
OUT      403C      0290
TXT      4063      0200 0330 0335


4000: 2A 5A 40 22 5C 40 CD 1F C0 CA 06 40 E6 7F 77 23
4010: FE 0A CA 31 40 47 CD 19 C0 78 FE 0D CC 4E 40 FE
4020: 7F C2 06 40 06 01 CD 19 C0 2B 2B C3 06 40 21 63
4030: 40 22 5A 40 2A 5C 40 44 4D C9 60 69 7E 47 FE 0A
4040: C8 CD 19 C0 23 78 FE 0D CC 4E 40 C3 3C 40 06 0A
4050: CD 19 C0 06 00 CD 19 C0 78 C9 63 40 63 40 00 00
4060: 00 00 00 00
```

The NOPs in the storage area are not necessary. I had them there to allow for quick 'patches' should the need arise. It also prevents destruction of the program should too many DELETES be accidentally entered. One of the changes that I am making in the new version is a check to make sure the user does not delete beyond the BEGinning of the current string being input!!

The BASIC5 sample program listing shows one way of implementing strings using this machine language program and CALLs. The user must first load this string handler using SOLOS. What I am doing at present is have my students write three short subroutines in BASIC up at the high end, say at 10000, 20000, and 30000. These subroutines contain the necessary CALL and ARG statements to access the string handler. This way, instead of trying to remember the addresses needed for the CALL statements, all the student need remember is that GOSUB 10000 inputs a string, GOSUB 20000 extracts a string, and GOSUB 30000 resets the string storage area.

I have further chosen to arbitrarily use Z as the variable name under which all ARG and CALL transfers take place. This simplifies writing BASIC programs using the string handler, since there is only one variable name to be remembered. For example, to input a string which is to store a person's name, you can simply say: GOSUB 10000: N=Z. This inputs the string and stores the address of the string in variable N. To recover this specific string, simply: LET Z=N: GOSUB 20000 and the string is printed out!



One caution: no leading and trailing spaces are imbedded into the string unless the user enters them himself. What this means is that if you do not provide such spaces yourself inside the BASIC PRINT statements that may surround the output strings, you may find that the string is printed with no intervening spaces, and that looks messy. If you find this a bother, then modify the program to add such spaces automatically. On the other hand, I use the fact that there are no spaces to good advantage in a game where the user puts in a bunch of technical words, and then the program combines them in various ways to form some long technical-looking, mind-bending words.

In any case, the program is simple enough to be easily expanded. I can't wait to get my hands on Processor Tech's 8K BASIC, but in the meantime at least I have a limited string capability to play around with. Incidentally, I find the string handler useful for programs other than BASIC. As with anything, the uses are as broad as the user's imagination! So imagine to your heart's content, and have fun! □

Continued from page 50.

pired during a system crash. I was puzzled until I remembered that the basis of the article was the notion that the EP enjoyed torturing the PSU. It seems that the EP enjoys this so much that it is willing to put itself through a great deal of trouble for such a tasty morsel.

What made this doubly curious is the description (with a picture yet!) of the strange garbage that the terminal prints when the system goes down . . . Here is the evidence folks! . . . Look what they do to us!

Mr Vallee is under the impression that we EP types have a great deal more control over what the computer does when the system crashes than I have ever witnessed. What to me is a disaster akin to a car accident is, to him, just EP sport. Evidently the computer should at least have the decency to type out, 'So sorry. But ill here. Be back presently.' as smoke curls up from the CPU or the read-write head digs a furrow through the disk.

SIMPLICITY REFUTED

I think that it is important to point out that computers are *not* simple. There is no conspiracy to make them seem complicated; they *are* complicated. The conspiracy is to make them seem simple to the terminal user. This illusion holds as long as everything works OK (just like your car). As soon as something goes wrong (the occasions Mr Vallee concentrates on) however, the thin veneer goes out the window and the terminal user may be dragged helter-skelter into the underlying reality.

THE AMERICAN WAY

Another notion expressed in the article is that the people in charge of satisfying the needs of the PSU are failing in their function to the extent that they fail to provide everything that the end user needs. This rather quaint idea is rooted in the notion of how American Capitalism is supposed to work. But is it the way that it does work or do you have to take your car to shop x to get the radio fixed, shop y to get the fender bumped out, shop z to get the wheels balanced?

It may be profitable to have someone check on individual terminal users and keep them supplied or it may be more profitable to let them fare for themselves and accept a few dropouts from the net. If, in fact, people shouldn't be dealt with in this manner then the problem should be addressed to the political and economic machinery rather than computer professionals.

LET'S BE FAIR

I realize that it was Mr Vallee's intent to deliberately present a one-sided terminal-user view of computers, but I wonder if, in moving the article from the original journal to *People's Computers* (which has a lot of readers whose contact with the computer is only through a terminal) a disservice hasn't been done . . . I don't know that further 'evidence' of EP evil doings presented to current terminal users in a simplistic manner serves any purpose.

It seems to me, also, that Mr Vallee's arrows are misdirected. Most of the things he complains about have more to do with economics, hardware failure, human fallibility, and the well-known difficulty of managing large software systems than 'programming', 'user science', sadists, or deliberate attempts at 'obfuscation'. □

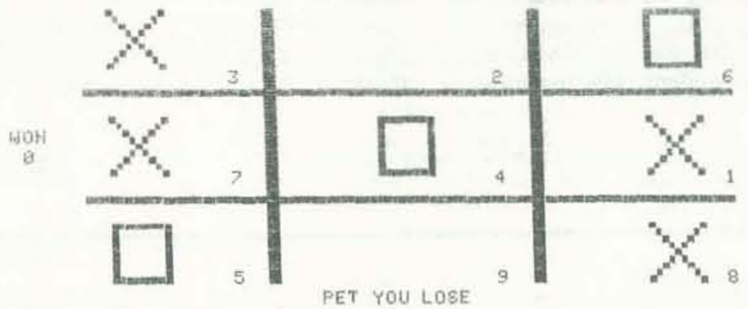
TRS-80 TALK

As you can see from this article, the TRS-80 has plenty of loyal fans as well as a fair share of critics.

Many thanks to Clyde Farrell for his TRS-80 Wumpus program. The Wumpus listing and run and the Tic-Tac-Toe game at the right were printed on Radio Shack's $599 TRS-80 screen printer at the recent Computer Faire in San Jose, CA. You press a button and whatever is shown on the video screen is printed (sideways) on a 4-inch wide strip of aluminum-colored electrostatic paper at a rate of 2200 characters per second.

The system shown at the Faire still had a few hardware glitches which caused dots to be randomly printed on the output; we 'cleaned up' the listings to improve readability.

In perusing the TRS-80 Catalog I noticed one ad that excessively annoyed me; unfortunately it's characteristic of many Tandy Computer ads. For $1198 you can buy the 4K 'Educator' System, which is nothing more than the standard TRS-80 with 4K RAM, Level I BASIC, video display, recorder and the screen printer described above. What I object to is the sentence '. . . the "Educator" is ideally suited for computer-assisted instruction programs'. As one who has been writing computer-assisted programs for 14 years, I can assure you that this is not *the case. Level I BASIC supports little of what most people associate with computer-assisted instruction, given its almost non-existent string handling capabilities and lack of file system. Such misleading advertising claims tarnish Radio Shack's image and are a disservice to those misled by them.*



Phyllis Cole, Editor

In response to your call for reports on the TRS-80, and also due to several 'negative' commments in your Jan-Feb issue, I am motivated to rally to Radio Shack's defense.

First, in Mr McCarthy's report, he mentions a $100 down payment required for a pig-in-a-poke machine. Perhaps the Radio Shack dealer he spoke with was ill-informed, but I was told that because there was not much information available on the TRS-80 at the time (about as much as there was on PET) a $100 *deposit* was requested that would be *completely* refundable if I was not happy with the product when it arrived (is this caveat emptor?).

Secondly, I ordered my TRS-80 with 4K of RAM but soon decided that 16K would be more to my liking. I changed my order (no problem!) and received my 16K machine at an increased cost of only $289 (are you listening Commodore?).

I have had my TRS-80 about a month now and have found that although Level I Basic appears to be limited at first glance, it has some 'hidden' capabilities that make it more attractive than a simple overview might reveal. Still, I am anxious to see what Level II can do for us.

Finally, I think it is commendable of both Radio Shack and Commodore that they have made the best (least expensive) contributions yet to providing computers for the average man. Bravo!! I look forward to seeing TRS-80 programs in the pages of *People's Computers* and would be delighted to submit a few myself. And thanks for *your* many contributions, long may they continue.

Clyde R Farrell
Walnut Creek, CA

WUMPUS

```
I FEEL A DRAFT
YOU ARE IN CAVE 17
TUNNELS LEAD TO CAVES 7  16  18
DO SOMETHING?S
CAVE NUMBER?18
MISSED
 3 ARROWS LEFT
*** EARTHQUAKE ***

YOU ARE IN CAVE 17
OOPS, YOU JUST FELL INTO AN UNDERGROUND POOL
DROPPED ARROWS !
 3 ARROWS LOST   0 ARROWS LEFT
TUNNELS LEAD TO CAVES 7  16  18
DO SOMETHING?_
```

The object of WUMPUS is to descend into a labyrinth of caves to hunt a WUMPUS and return to the surface with your catch, while coping with the many hazards that befall you during your adventure. In this version, each turn you may

1. Proceed. . . to a new cave.
2. Shoot. . . into a connecting cave.
3. Count. . . the number of arrows that you have.
4. Exit. . . from the caves *if* you are in the exit cave.

Level I BASIC lets you assign a value to a variable and then later use that variable as a numerical input. This is why you can respond with 'P' for Proceed instead of typing '1', as 'P' was assigned a value of '1' in line 2475. This makes the game more enjoyable because you don't need to remember what number means what command!

Level I BASIC does not support 2-dimensional arrays, but I've 'simulated' them using the 1-dimensional array in my WUMPUS game. I calculate the correct index for the 1-dimensional array by using the *second* parameter of the array as a multiplier for the first parameter, and then adding the second parameter back in. For example, if an array in a program is dimensioned as A(20,3) and you were looking for the data contained in A(J, K), you would look in A(3*J+K). So A(4, 2) is A(3*4+2) or A(14), in our single dimension array. This idea is further exemplified in line 70 where our 'two-dimensional' array is filled with the required data.

Line 4210 shows Level I BASIC's method of using Boolean logic; '+' means 'OR', '*' means 'AND'.

A(101) is the cave you are in.
A(102) is where the WUMPUS is hiding.
A(103) and A(104) are caves with bottomless pits.
A(105) and A(106) are caves containing superbats.
A(107) is a blocked cave.
A(108) is the exit cave.

Level I abbreviations used in the listing are:

RET.=RETURN	IN.=INPUT	N.=NEXT	G.=GOTO
GOS.=GOSUB	P.=PRINT	F.=FOR	T.=THEN

Also, spaces have been deleted to conserve memory.

WUMPUS and other programs, including STAR TREK, are available for LEVEL I users through

Farrell Enterprises
PO Box 4392
Walnut Creek, CA 94596

```
   5  CLS
  10  P."WELCOME TO 'HUNT THE WUMPUS'":P.
  15  Y=1:N=0
  70  F.J=1TO20:F.K=1TO3:READA(3*J+K):N.K:N.J
 130  DATA2,5,8,1,3,10,2,4,12,3,5,14,1,4,6
 140  DATA5,7,15,6,8,17,1,7,9,8,10,18,2,9,11
 150  DATA10,12,19,3,11,13,12,14,20,4,13,15,6,14,16
 160  DATA15,17,20,7,16,18,9,17,19,11,18,20,13,16,19
 240  W=0:F=W:M=W:L=100:F.J=1TO7:A(L+J)=RND(20):N.J
 280  F.J=1TO7:F.K=JTO7:IFJ=KT.330
 320  IF A(L+J)=A(L+K)THEN240
 330  N.K:N.J:A=5:A(L+8)=A(L+1):P.:P."ENTRANCE IS IN CAVE";A(L+8
0
 390  IFRND(10)>4GOS.3370
 485  IFRND(100)<8GOS.5000
2000  P.:F.K=1TO3:F.J=2TO6:IFA((A(101)*3)+K)<>A(L+J)T.2110
2050  ONJ-1G.2060,2080,2080,2100,2100
2060  P."I SMELL A WUMPUS":G.2110
2080  P."I FEEL A DRAFT":G.2110
2100  P."BATS NEARBY":G.2110
2105  P."I SEE DAYLIGHT!!"
2110  N.J:N.K:P."YOU ARE IN CAVE";A(L+1)
2140  J=INT(RND(0)*40):IF(J=0)+(J>7)THEN2440
2150  ONJGOS.2200,2210,2220,2220,2230,2240,2250:G.2440
2200  P."AHA!..WUMPUS TRACKS!!":RET.
2210  P."AHA!..FOUND AN OLD ARROW, LUCKY YOU":A=A+1:RET.
2220  P."OOPS, SLIPPED ON SOME LOOSE GRAVEL":G.5900
2230  P."OOPS, YOU JUST FELL INTO AN UNDERGROUND POOL":G.5900
2240  P."THIS LOOKS LIKE A NICE CAVE, LET'S STOP FOR LUNCH":RET
2250  P."TAKE CARE WITH THAT FLASHLIGHT!!":RET.
2440  P."TUNNELS LEAD TO CAVES";:F.Q=1TO3:P.A(A(101)*3+Q);:N.Q
2450  IFJ=38THENQ=A(105):G.4260
2475  S=1:C=2:P=3:E=4
2480  Q=0
2500  M=M+1:P.:IN."DO SOMETHING";Q:IFQ=STHEN3000
2550  IFQ=CTHEN3220
2560  IF(Q=E)*(A(101)=A(108))THEN8800
2570  IFQ=PTHEN4000
2590  G.2500
3000  IFA<1P."WHAT WITH? ..DUMMY":G.390
3010  IN."CAVE NUMBER";Q:F.K=1TO3:IFA(A(101)*3+K)=QTHEN3130
3020  N.K:P."NOT POSSIBLE":G.3010
3130  A=A-1:IFA<0THENA=0:G.3220
3135  IFQ<>A(102)THENP."MISSED":G.3215
3140  IFRND(10)<7T.P."YOU GOT THE WUMPUS":F=F+1:A(102)=0:G.4400
3150  P."YOU WOUNDED THE WUMPUS"
3213  GOS.3370
3220  P.A;"ARROWS LEFT":G.390
3370  IFW=1THENA(102)=0:RET.
3380  A(102)=A((A(102)*3)+RND(3))
3385  IF(A(101)=A(102))+(A(102)=A(107))T.3380:RET
3425  RET.
4000  IN."WHERE TO";Q:F.K=1TO3:IFA(A(101)*3+K)=QTHEN4120
4080  N.K:IFQ<>A(101)T.P."NOT POSSIBLE":G.4000
4120  IFQ=A(102)P."OOPS! BUMPED A WUMPUS!":G.4500
4210  IF(Q=A(103))+(Q=A(104))P."YYYIIIIEEE...FELL INTO PIT":G.4
520
4220  IFQ=A(107)P."CAVE ENTRANCE IS BLOCKED":GOS.5900:G.390
4260  IF(Q=A(105))+(Q=A(106))P."ZAP...SUPERBAT SNATCH!":G.4280
4265  IFQ=A(108)P."EXIT NEARBY"
4270  A(101)=Q:G.390
4280  Q=RND(20):IF(Q=A(101))+(Q=A(106))+(Q=A(107))THEN4280
4290  GOS.5900:G.4120
4400  IFRND(100)<75P."BEWARE OF IT'S MATE!!":J=2:GOS.6100:G.322
0
4410  W=1:P."HEE HEE HEE...THE WUMPUS'L GET YOU NEXT TIME":G.39
0
4500  IFRND(100)<75GOS.3370:GOS.5900:G.4270
4510  P."TSK TSK TSK...THE WUMPUS GOT YOU"
4520  P."HA HA HA...YOU LOSE":G.8810
5000  P."*** EARTHQUAKE ***":F.J=3TO7:GOS.6100:N.J:GOS.5900
5005  A(102)=RND(20):IF(A(101)=A(102))+(A(102)=A(107))T.5005
5010  IFRND(10)>1T.RET.
5020  A(108)=RND(20):F.J=3TO7
5030  IFA(108)=A(L+J)T.A(108)=RND(20):G.5030
5040  N.J:RET.
5900  J=RND(10)-1:IF(J>A)+(A=0)T.RET.
5910  P."DROPPED ARROWS!!":IFJ=0P."ALL ARROWS FOUND":RET.
5930  A=A-J:P.J;"ARROWS LOST",A;"ARROWS LEFT"
5931  RET.
6100  A(L+J)=RND(20):IF(A(L+J)=A(101))+(A(L+J)=A(108))T.6100
6200  RET.
7730  P."OR IF YOU RETURN TO THE ENTRANCE CAVE YOU WILL BE"
8800  P."OUT OF THE CAVES   ",:IFF>0P."GOOD HUNTING"
8805  W=F*1000/M:P."YOUR RATING IS";W
8810  IFF=0P."BETTER LUCK NEXT TIME"
8815  Y=1:N=0:IN."WOULD YOU LIKE TO TRY AGAIN";Q
8820  IF Q=YT.CLS:G.240
9000  END
```

The January-February issue of your publication indicates you've had some bad experiences with the Radio Shack TRS-80 and solicits users' comments. Well, here's mine. I've been enjoying my TRS-80 for several months, and the one time I needed it, got excellent service at the Radio Shack repair center in Belmont, CA. The current software is unbelievably primitive compared to the PET's, but with the new software announced this week, that situation will probably be changed.

As a learning machine I find the TRS-80 excellent. I still haven't finished writing all the possible programs and I'm sure I won't by the time the Level II BASIC arrives. The book leaves a lot to be desired. But that can and should be remedied by someone (you? me?) writing a better book.

As a start, I am contacting anyone anywhere who advertises a users' group for the TRS-80. I will probably attempt to start a group soon myself, if my busy schedule allows the time. And I'll soon have programs available, with complete printed instructions and documentation, at about the same price as Radio Shack. I can now offer documentation on the Radio Shack BASIC programs I have.

The neighborhood kids call and almost literally stand in line for a chance to use the TRS-80, and I find it a lot easier to use with its almost standard typewriter keyboard layout than the PET with its small keys. My youngest operator-programmer is only 7, and smart enough to use the level I BASIC. As and when I can get a PILOT assembler or enough BASIC to try the BASIC PILOT in one of your issues, I'll have even more of the younger set around, I'm sure.

All in all, I find my 16K system (with no heat problems by the way, as the 4K version has) a very good buy for the money, a very good chance for the average beginner to get into microcomputers, and a lot of fun. The graphics, even in the Level II, are not as good as the PET's, but I need a usable keyboard much more than fancy graphics. (Any truth to the rumor heard today that PET is no longer being distributed?)

Jeff Lasman
San Mateo, CA

8K PET's are alive and well and even available off-the-shelf in some Northern California stores. Production of 4K systems has been discontinued, at least for now.

The file system for the TRS-80 Level II BASIC is improved over the first version; it is no longer necessary to unplug cables to rewind tape. All tape positioning controls (tape start, stop, rewind, etc) are under manual control. Named files can be written and read from tape without manually positioning to the beginning of the tape with one curious exception: when a new tape is put into the cassette drive, it must be manually positioned so that no leader is showing. The Radio Shack salesperson at the Faire said that Radio Shack was going to put out a line of leaderless tapes. This is plainly the wrong fix for the problem; it gives the poor user the choice between non-standard tapes or the manual operation. The right fix is to redesign the cassette controller so that it works with unmodified audio cassettes.

The bad news is that the names of the named files are limited to one character; the universe of available names is thus quite small. File read/write status is indicated by a blinking/stationary asterisk notation in the upper right corner of the screen. The single character file name also appears, but apparently only while the file is being written or read. No history of files previously encountered is preserved on the screen. This is unnecessarily cryptic and clumsy regard for human factors, especially in a machine intended for naive users.

Dave Caulkins
Los Altos, CA

I have a number of comments about the TRS-80. These are based on a few weeks of intensive fiddling around with the same machine that *People's Computers* used for their review. However, before it got to me the transformer blew and it went back to the factory for repairs.

Hardware: The keyboard is fine. It lacks rollover, but being only a fast hunt & peck typist I wasn't really bothered. I liked having the keyboard separate from the CRT but I found all the power cords a nuisance. The CRT was adequate. I had no trouble with the cassette recorder at all: not a single error in several dozen LOAD and SAVE's.

System Software: Mediocre. Also rather slow. As a test, I ran the benchmarks that Feldman & Rugg used for their *Kilobaud* article (issue No 10, Oct '77, pages 20-25) on timing comparisons. The times were in seconds: 2.5, 18.0, 34.0, 39.0, 45.5, 67.0, 110.0. That puts the TRS-80 with Level I BASIC number 25.5 on their list. A bad showing for a Z-80 machine.

As with many machines, the advertised amount of memory is not the usable amount. The 4K version of the TRS-80 has only 3½K for the user (3583 bytes). This is good for about 100 lines of BASIC depending on how much array space you need, how much you use multiple lines, and whether you use abbreviations.

Interestingly enough, the BASIC looks like good old Palo Alto Tiny Basic with a few bells and whistles. The string capabilities aren't worth two cents as far as I'm concerned. It does allow point plotting but this feature is as slow as the rest, if not slower.

Documentation: No real comment here. For anyone who already knows BASIC it shouldn't take more than half an hour to extract everything you need from the manual.

In General: I wouldn't recommend the TRS-80. While it does work and is reliable, I don't consider that sufficient. The system software is mediocre—a bad mark for a machine intended to be self-contained. Overall, I could find nothing exceptional about it. It doesn't do anything better than other machines and it really doesn't do as much.

Eryk Vershen
Palo Alto, CA

I have owned a TRS-80 for a month and am convinced the product as a whole is superior to anything else on the market. I can think of four reasons right away. First. Radio Shack is indeed delivering their TRS-80, as advertised, and is already following through with a goodly number of upgrading products. The company doesn't demand cash-in-advance and it doesn't go seeking publicity until it is ready to fulfill the expectations it raises. I care strongly about this: I waited four and a half months on a Commodore PET order and received nothing but a defensive letter from a marketing vice-president. Radio Shack is actually fulfilling the promise their competition has made: an affordable computer mass-produced for personal use.

Second, the TRS-80 has the most extensive dealership network of any microcomputer. The typical Radio Shack dealer knows little about the product he's selling, but he's courteous and willing to help in any way he can. He's *available*, and few micro dealers have his resources.

Third, the machine itself works very reliably in my experience. It's been quite a capable system from the moment I plugged it in. Certainly Level I BASIC is not a business language, and I'll get Level II ASAP, but it's sure got the edge over machine language and the Tiny BASIC of last year. With all the hardware and software products already announced, I feel very well supported.

Fourth, Radio Shack's user's manual is excellent! It takes a novice owner step-by-step through a pretty good first programming course, and does it gently and pleasantly. All too many people think of computers as difficult and intimidating, and this author reveals the fun and simplicity that is the essential core of all learning.

So, with reasonable delivery, so many dealers, a complete and reliable system, and *such* a good instruction manual, why do you people have such long faces? In my opinion the TRS-80 is no less than revolutionary!

Mark R Johnson
St Louis, MO □

DRAGONSMOKE

BY THE DRAGON

The Dragon, sometimes known as Bob Albrecht, was the founder of this periodical way back in 1972. He also edited it for its first four years until yours truly took over with Volume 5, Number 3. Bob has spent the last few years working with kids, computers, and calculators in schools. He's gotten very interested lately in fantasy games, and will continue to share ideas about them in future issues.

Phyllis Cole

So! Last issue you read 'Epic Computer Games' by Dennis Allison and Lee Hoevel. You are hooked—you want to play or perhaps even write an epic game. In case you don't already know where to collect information on role-playing fantasy adventure games, here are some info sources.

TSR Hobbies, Inc.
P.O. Box 756
Lake Geneva, WI 53147

TSR invented *Dungeons and Dragons*. Try one or more of the following.

- **DUNGEONS AND DRAGONS.** The basic game—dungeon geomorphs, monsters, treasure, polyhedra dice and the D & D rule book for levels 1 to 3. $9.95 + $1.00 postage and handling.
- **DUNGEONS.** A highly-simplified board game version of D & D for 1 to 12 players. I've played it with kids, 8 years old and up. $10.00 + $1.00 postage and handling.
- *THE DRAGON.* TSR's magazine of swords and sorcery, fantasy, and science fiction gaming. Monthly, $18/year.

THE CHAOSIUM
P.O. Box 6302
Albany, CA 94706

- **WHITE BEAR AND RED MOON.** A board game in which you are the ruler of a legendary army during the battle of Dragon Pass. $9.95
- *ALL THE WORLD'S MONSTERS,* edited by Jeff Pimper and Steve Perrin. A compendium of monsters to populate your fantasy adventure worlds. Two volumes—350 monsters in Volume 1, 250 monsters in Volume 2. $7.95 each.

METAGAMING
Box 15346
Austin, TX 78761

- **MELEE.** A folio game of man-to-man combat with archaic weapons. $2.95
- **WIZARD...** the magical combat system, a game of magical duels for two or more players. $3.95
- **MONSTERS! MONSTERS!** A fantasy game for the bad guys, in which monsters get equal time. $7.

For more information, find a hobby shop that specializes in fantasy games. I collected the stuff on this page at:
Outpost Hobbies
224 California Drive
Burlingame, CA 94010
And—watch DRAGONSMOKE for more Dragon Data. □

BY PHYLLIS COLE

EDUCATIONAL SOFTWARE

Recently I've gotten involved in distributing software (as a volunteer) for an increasing number of hours per week. So I'm looking at potential distributors who will distribute the materials in exchange for paying a royalty to the school that holds the copyright on the materials. Many would-be distributors of software for home computers showed up at the recent Computer Faire in San Jose. They all had one thing in common: they realized that the field was potentially a lucrative one, but had few ideas about how to go about exploiting it. Most potential distributors had some sort of vague proposition to make, immediately followed by 'How does that sound? What do you suggest?' Those questions led me to try to concretize my ideas about what I, as a freelance author of software, would like to see offered by a distributor.

My concerns are biased towards educational software, in part because that's the field in which I expect to be writing. However, I also believe it is in the area of educational software that the potential of personal computers may truly be realized. By the way, I define education as broadly as possible—many video games are educational.

Our educational system is simply not doing the job that many of us want it to; more and more parents and students are finding that the majority of learning takes place not in the traditional classroom, but in more informal ways, such as building electronics kits, parttime jobs, etc. With this realization has come a hunger for personalized educational materials both for the classroom and the family room. And the image of the home computer as a potential private tutor comes to mind.

Possible goals for a distributor of educational software might include:

* developing the company's reputation of having a 'seal of good educational software' on all its products

* providing classroom-tested programs and classroom-support materials at reasonable prices

Products would consist of one or more educational programs available on cassette tape. Several differing support packages could be offered—one minimal one, another for the 'family room educator' and a third for typical classroom use. The programs should cover topics suitable for students of all ages—adult education is an area that looks particularly interesting. Nor should materials for very young children be ignored; systems with graphics capabilities can be used to produce a variety of pre-math and pre-reading picture-oriented games and exercises.

Products initially should be developed for systems whose projected sales are on the order of 50,000-100,000 systems per year. Marketing should be directed at both home and school. Evidence that owners of home systems are interested in educational software comes from results of a recent readership survey for *People's Computers*: about 33% of those responding to the survey identified themselves as educators, but 76% of those replying expressed a desire for educational software.

Already educational publishers are distributing reading programs based on cassette tapes supported by workbooks, etc. Royalty payments are already established in the field as a way to attract and reimburse authors. Various companies are tooling up to mass produce computer software on cassette tapes, with the needed quality control.

Pricing should take into account that reasonably priced programs have the best chance of not being ripped off. Another way to avoid the rip-off problem is to make documentation so useful that the purchaser is inclined to buy the reasonably-priced and easily available product rather than go to the trouble of reproducing the documentation.

Reasonable royalty payment to authors of software are essential if high quality programs are to be produced on an on-going basis. For thoroughly documented programs, the standard 10-15% of retail price traditionally offered as a royalty by textbook publishers seems fair.

The key to the future of the home/school computer rests on the quality of the software and documentation that will be produced for the systems. The hardware problems are being solved at a pace far exceeding that of software problems. It remains to be seen whether quality programs and documentation that appeal to consumers can be produced and distributed. The potential is there: computers can help fill the demands heard from all segments of society for better education and re-education for people of all ages. Authors are beginning to appear with some very interesting materials; hopefully the kind of software distributors we need will soon materialize. □

A Call for Distributors

APL WANTS YOU!

BY MOKURAI CHERLIN

Reverend Mokurai Cherlin is a Buddhist Priest who moonlights as a programmer for his father's company, APL Business Consultants, Inc. He has done all his programming so far on an Amdahl 470 and hopes to get on an IBM 5100 sometime, and on any microcomputer that has APL as soon as it comes out. It should be clearly understood by all that he has no intention of writing anything called Zen and the Art of Computer Programming

With the recently released FORTRAN IV compiler and the forthcoming APL interpreter for microcomputers, both from Microsoft, it can be said (again!) that real computing power is now, if not in the hands of the people, at least available to them. Soon it should be possible to buy a real full-power computer off the shelf with the capabilities of the IBM 5100 portable computer and a price tag under $1500. The 5100, priced at $9000, has built-in cartridge I/O, about 100K of memory, and a few other goodies. The $1500 machine will provide about the same capabilities at one-sixth the cost of the 5100.

THIS MEANS YOU

To many of you, news of APL for micros does not seem exciting or even interesting, because APL has unfairly gotten the reputation of being difficult to understand, usable only by mathematical wizards, and expensive in terms of memory and time—and it makes Bob Albrecht's teeth rattle. Experience has shown that these opinions are greatly exaggerated. The experience of IBM itself is the clearest case. APL was developed by mathematician Ken Iverson. When APL was first implemented in the late 60's, IBM did not think they would be able to sell APL to anyone. IBM did implement APL on their machines for use in experiments on various aspects of their operating systems. In order to get meaningful results, they had to have a normal user load of real work, so they let their employees use the APL system as much as they liked, for everything from one-line calculations to hours of number crunching.

The results amazed IBM and made them release APL as a program product: thousands of their programmers switched over to APL and wouldn't go back. Even more amazing, thousands of people who couldn't or wouldn't learn programming before picked up APL and loved it. Many of them wrote significant applications in the first week, even those who had never done any programming before.

Now one may well ask what can make a language so attractive that it makes converts of people who have resisted IBM's best efforts to interest them in programming. What are more than 15,000 people using at IBM that we don't have? Why don't we all know about this, we who are so eager, perhaps even desperate, for tools which will let us bring computers to the masses?

We don't have it simply because it has been too expensive for us, with timesharing at $20/hour or more. The new interpreter from Microsoft will go a long way toward bridging that gap, since it will run on any 8080 or Z-80 based system with 24K for the interpreter and 8K-40K to work in. The reason we don't know about it also results from the expense incurred by needing a minimum of 32K of memory.

From the outside, APL can be intimidating; it only reveals its power and convenience in actual use, as IBM found out. Just to list the features of APL would take more room than I have, and would still not give the real feel of the language. There is no substitute for getting on-line and messing around with it.

USER ORIENTATION

The particular virtue of APL from the point of view of the frustrated learner or teacher is the fact that one can get on the system and play with it, learning by doing, without having to know any more than how to sign on and off and how to load workspaces. A workspace is like a page in a notebook. Workspaces can be named and loaded selectively; some are public, others are private.

There is no known way to make the system crash; any attempt to go beyond the limits of the system results in an error message, and the user can then try something else. When an error is found in the middle of a function such that execution cannot proceed, the state as of the last completed statement is saved, and the location and nature of the error are printed out. The user can then examine variables, run diagnostics, rewrite the function, and either continue from where he left off, run the program over again from the beginning, try any other program, or force an exit from the suspended program.

Most accounts of APL power concentrate on the built-in functions and the ability to do vector, matrix, and higher dimensional array operations directly without program loops. For many users this is the most impressive part of APL power, and anyone who has had occasion to invert a matrix will appreciate having a function that performs this operation with one symbol, ' ⌹ '. People who have had to give up a project or not start one because such a function was lacking will appreciate it even more.

But this is not all that makes APL desirable, especially to those with no interest in mathematical applications. (I don't want to minimize the importance of powerful mathematical functions, either. Until you have a convenient form of some tool, you may not know how much you have always wanted it.) The value to the non-specialist comes particularly from the convenience of knowing immediately how you are doing, and having understandable help in doing something about it. The literature of learning has pointed out in great detail the importance of immediate feedback, and every teacher has seen all too often the ill effects of frustration and delay on students' interest and ability to learn.

No one should suppose that APL will correct all mistakes itself or give them cleaner white teeth. What it will do, to a greater extent than other languages, is let the user get to work. There is no problem with duplicating variable names in subroutines, since variables outside the function can be shielded. It is not necessary to keep track of numerous parameters because so much looping is eliminated; subroutines required in other languages can frequently be replaced by primitive APL functions. Much larger and more complicated problems can be tackled because APL programs are commonly one-tenth the length and complexity of FORTRAN or BASIC programs for the same amount of processing (yes! you can write short readable lines of APL). In short, you can get on with solving the problem and spend less time coding and keeping track of trivia, by letting the computer take care of much of the drudgery for you. Computers do all that much better anyway.

To explain APL in any detail requires a book. Anyone who is interested in reading about the language should get either *A Programming Language*, by Kenneth Iverson, the original source of the language and the acronym, or *APL: An Interactive Approach*, by Gilman (IBM) and Rose (Scientific Time Sharing Corporation–STSC). The latter is a textbook which guides the learner through the language on-line, and can be used off-line since all examples are illustrated with actual terminal printouts. Both are available from STSC and many computer stores.

FEATURES

One of the prominent features of APL is the variety of input modes: immediate execution, function definition, evaluated input and string input. In the immediate execution mode, whatever is typed is carried out when you press carriage return. These examples show some uses of APL in immediate execution mode.

APL looks quite conventional when we perform a simple addition:

```
      2+2
4
```

However 3*2+4 evaluates to 18, since evaluation is right to left without precedence. We can perform a decimal to octal conversion by typing '8 8 8 ⊤ ' followed by the decimal number to be converted; APL responds with the octal number.

```
      8 8 8 ⊤ 76
1 1 4
```

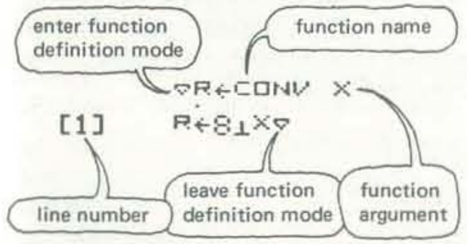
Similarly, we can convert from octal to decimal. Type 8 ⊥ (ie the inverse of 8 8 8 ⊤) then the octal number to be converted. APL prints 76, the decimal equivalent in this example.

```
      8 ⊥ 1 1 4
76
```

Here's the type of response APL gives when you try to perform an illegal operation–in this case, dividing by 0. Note that in the fourth line the ' ^ ' indicates the ÷ operand is the source of the problem.

```
      5÷0
DOMAIN ERROR
      5÷0
      ^
```

Function definition mode allows functions to be written for later execution, or rewritten at any time. The del character, ∇ , is the signal to enter or to leave function definition mode. Here's an example of a one-line function for octal to decimal conversion.



```
      ∇R←CONV X
[1]   R←8⊥X∇
```

enter function definition mode
function name
line number
leave function definition mode
function argument

Next we try out our function, CONV. And we find it works–114 octal is 76 decimal.

```
      CONV 1 1 4
76
```

The quad character, ⎕ , allows numeric and character input and output in the middle of execution, as shown below.

```
      CONV X←⎕
⎕:
      1 1 4
76
```

The computer requests data input, and then executes the remainder of the line. Quad input is evaluated before being handed to the functions which will operate on it; it can therefore be entered in any legal APL expression: numbers in any format, function calls, variable names, and file references among them.

We can edit CONV and replace the argument X by a quad. Here's what happens when our re-defined CONV is called.

```
      CONV
⎕:
      1 1 4
76
```

The quad accepts any APL expression as input, so in the above example, we could write 100 + 14 as input and get the same result.

Quote-quad, ⍞ , accepts a character string in a manner similar to quad but without evaluating it. Quote-quad rejects illegal characters with a request to try again. Quad will accept a character string with quotes around it as data, and quote-quad will accept a string without quotes so that one can simply type the appropriate word, statement or what have you without bothering about format.

Next we demonstrate using more than one statement on a single line. A diamond, ◇, is used to separate statements which will be executed in sequence. The first statement, 'X:' enters X: as a character string in immediate mode, which causes the string to be typed out. The second statement is ⌽ ⍞ ; the quote-quad accepts our string input, and the function ⌽ reverses the input string.

```
              'X:' ◇ ⌽⍞
      X:
      OLLEH
      HELLO
```

The structure of the APL system provides capabilities that must be seen to be appreciated. There are many powerful operators and many system functions which allow for extreme flexibility in operation. Character data can be converted to functions and executed, the character array can be brought in from any available source; input can be through quad (numbers or expressions) or quote-quad characters, files, variables, and function values. A function can define another function or edit one already defined, then convert it to character form and store it in a file or use it as a variable. It can turn a stored function into an active one and call it, and so on and on.

This is where the real power of APL resides. A set of functions stored in a file as character strings or matrices can be called up and executed in turn under program control, even though only one of them may fit in the workspace at a time. The same effect can be produced in another way by putting each function in a different workspace, so that each workspace can call its own function, store its results in the file, and call the next workspace.

One of the features which makes this possible is called the latent expression. A workspace can be stored with any one-line expression set to execute immediately as soon as the workspace is loaded. The latent expression can print instructions and call the main function in a tutorial program, so that the user need only know how to sign on and load the workspace.

There are many features of APL that I have not mentioned at all, or have only barely touched on, such as security provisions, output formatting, and the compound operators whose arguments are primitive APL functions and whose results are other powerful functions. But perhaps there will be another opportunity to write on these and others.

SHORTCOMINGS

By now it should be clear that I am a true believer. Nevertheless, I am aware of shortcomings in APL. The error diagnostics could be made much more informative; editing facilities could be expanded; some improvements in the debugging facilities could also be made. The chief difficulty with APL is space. The interpreter is large by current micro standards; a workspace with nothing in it takes up 4K for stacks and tables. The price of memory is still tumbling down at 30 to 40% a year, and lots of bright and industrious people are busy writing improvements and enhancements for APL all the time, so relief is in sight in all of these matters.

GO TO IT

I don't expect to make believers of all who read these words. There is no question that APL is formidable when first approached. If I have gotten you interested, I urge you to find an APL system and get some experience with it yourself. IBM is happy to demonstrate the 5100 and 5110 to anyone who looks like a customer, even if only for cartridges or paper. STSC is equally eager to show off its APL*PLUS ® system to anyone who might be interested in buying time, programs or books from them. They also sponsor free courses and workshops for actual and potential users. See your nearest big city phone book under 'data processing' for offices of both. When Microsoft's interpreter gets into the computer stores there should be no trouble getting a demonstration and a tryout. So get on and get hooked! □

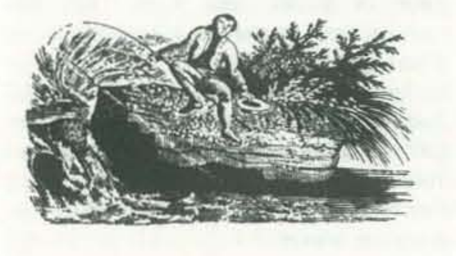
ANNOUNCEMENTS

16 PORT SERIAL BOARD

Ohio Scientific announces its 16 port serial I/O board. This board is available for use on any Ohio Scientific computer system. It comes fully assembled as CA10-X where X specifies number of serial ports on the board from 2 to 16. The board features RS232 and high speed synchronous interfaces which can be mixed in any combination. The communications transfer rate of each serial port is jumper selectable from 75 to 19,200 baud asynchronous or 250 to 599 Kbits in a synchronous mode. Each port is based on a fully programmable ACIA which is capable of running both the asynchronous or fully synchronous. The interface board is available as a CA10-X for $200 retail for the first two ports plus $50 additional for each extra port up to 16. Contact Ohio Scientific Industries, 1333 Chillicothe Rd, Aurora, OH 44202; (216) 562-3101.

FULL ASCII KEYBOARD

The Model 756 full ASCII Keyboard provides encoding for all 128 ASCII characters and control functions. The 756's line of accessories includes a numeric pad, custom cables and connectors. The interface allows user selection of parity, positive or negative logic data and strobe outputs, alpha lock operation and both D.C. level and pulse strobe signals. A latching shift lock key is included, and all outputs are TTL-DTL-MOS compatible. The 756 is available in either kit form or assembled and tested. Retail price for the Model 756 kit is $64.95, and assembled and tested for $75.95. Contact George Risk Industries, Inc, G.R.I. Plaza, Kimball, Nebraska 69145; (308) 235-4645.



PET-488 BUS CONNECTOR

The PICKLES & TROUT PET-488 cable assembly makes your PET Computer plug compatible with any IEEE-488 device. The inexpensive PET Computer can thus become the controller for a wide variety of electronic test equipment and computer peripherals that can talk to the IEEE-488 bus. The cable itself meets all specs for shielding and cross-talk and is 18 inches (.45m) long. Price is $30. Contact PICKLES & TROUT, PO Box 1206, Goleta, CA 93017.

MAILING LIST SOFTWARE

This modular mailing list package sorts on zip code or title address, merges files or extracts sub-files, and prints envelopes and multiple-column labels. The complete software is $75 on a single density CP/M diskette, in either Microsoft BASIC or Commercial BASIC. Contact the Center for the Study of the Future, 4110 N.E. Alameda, Portland, OR 97212; (503) 282-5835.

MAILING LIST PACKAGE

The Comprehensive Mailing List Package #ML-1NS enables the user to start and effectively maintain one or more mailing lists. Operations include: Add, Delete, Search, Sort, Auto-Sort, and Sequential Printout. Features include: user-selectable defaults for ease of entry, user-selectable number of labels across page for different printers and label sheets, and user-selectable 3 or 4 line address for each independent entry. The program set is written for convenience and ease of use. Available with complete documentation and North Star diskette for only $25 PPD. Delivery is from stock. Documentation package only is $4.50 PPD, fully refundable with order for diskette. A SWTPC disk version will be available soon. Order from: Williams Radio and TV, Inc, Computer Division, 2062 Liberty Street, PO Box 3314, Jacksonville, Florida, 32206.

FLOPPY FILE SYSTEM

KSAM is a file management system designed specifically for floppy disk microcomputer systems. Random storage and retrieval of records is based on the contents of a user-defined data field within the record which is called the key. The system supports sequential access of records starting at any point within a file, random access by partial key and random access by relative record number. Sequential and random access commands can be intermixed freely.

Space is automatically allocated to the file when records are added, and reclaimed when records are deleted. KSAM80's buffering techniques make the average retrieval time for any record significantly less than the time required to perform the same access by track and sector address. A number of utility programs are available as part of the KSAM80 package.

KSAM80 was originally developed under Zilog's Z80 OS 2.0 but can be easily implemented in many existing microcomputer operating systems. For additional information or personal demonstration contact EMS, 3645 Grand Ave, Suite 304, Oakland, California 94610; (415) 834-4944.

BUSINESS SOFTWARE

This business package includes a General Ledger package, an Accounts Receivable, Accounts Payable, and Payroll package, an Inventory and Manufacturing package, and a Mailing List package. Features include the ability to print a variety of checks, invoices, purchase orders, and mailing labels. Required equipment includes a line printer (Okidata 22 preferred), a terminal (Soroc IQ 120 preferred), Dual North Star disk drive system with North Star BASIC and 32K memory. The $295 package is available from Aaron Associates, PO Box 1720A, Garden Grove, CA 92640; (714) 539-0735.

STAR WARS SIMULATION

The Star Wars program from Objective Design is a true, real time simulation. Under player control, ships move in three dimensions to create a realistic simulation of actual space flight. Objects increase in size as the ships approach and diminish as they pass. Weapons, deflector screens, and a directional control joystick are implemented in each ship. True to the original storyline, ships of the Rebel forces must pass through Imperial defenses and Tie-fighters to enter a channel on the Death Star. If they can avoid a crash into the channel wall and avoid the gunsights of pursuing ships, they have a chance to destroy the Death Star.

The game requires the high density graphics display provided by Objective Design's Programmable Character Generator. This S-100 card can be used with the Processor Tech VDM or SOL, Polymorphic Systems VTI, Solid State Music Video Board, and other video boards using the Motorola family of 9 x 7 matrix generators, and sells for $169.95 kit, and $215.95 assembled. Written in 14K of 8080 assembly language, the program code is being offered on Tarbell and CUTS tape. Game rules and instructions for assembling the required ship control boxes are included in the total price of $7.50. Contact Objective Design, Inc., PO Box 20325, Tallahassee, FL 32304; (904) 224-5545.

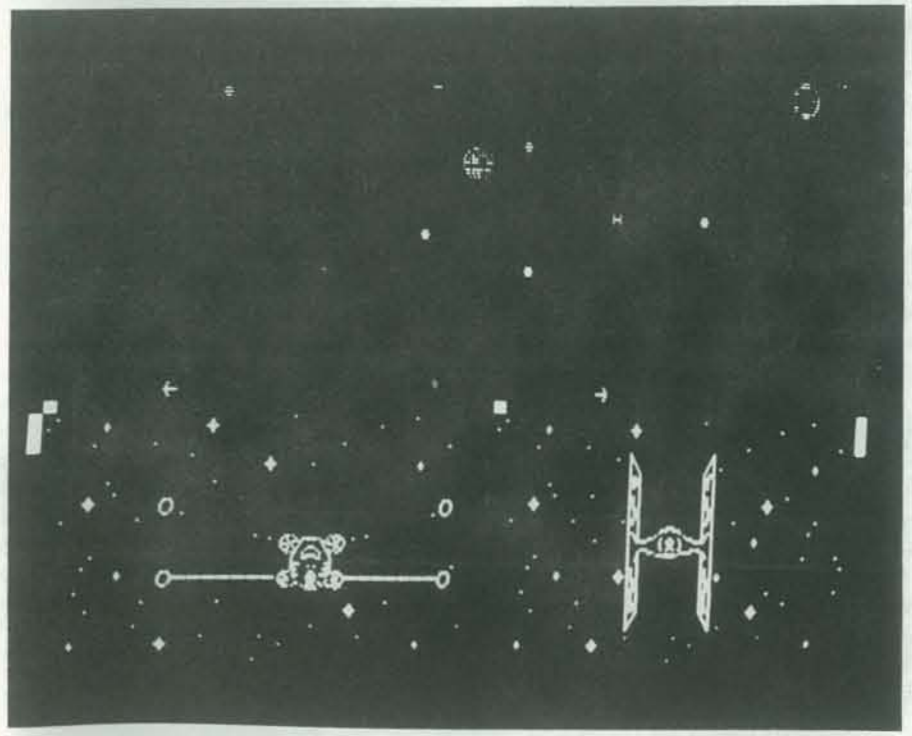
APPLE GOES TO SEED

The 'Apple Core' is the new San Francisco Apple users' group. To qualify as a member of 'The Apple Core' you must own or regularly use an Apple in any memory configuration. You must also pay dues, the amount of which is yet to be established.

Sorry to make the membership requirements so tough, but we gotta keep the riff-raff out some way (would you want an Altair to move in next to you?). Contact Scot Kamins, SF Apple Users' Group, Box 4816, Main Post Office, San Francisco, CA 94101.

TRS-80 USERS GROUP

The TRS-80 Users Group of Eastern Massachusetts expects to be a popular and useful clearinghouse and generator of activities concerning effective use of the TRS-80. It solicits information on all TRS-80-compatible hardware and software. Interested TRS-80 users are invited to attend meetings, held 7:30 p.m. on the second Wednesday of each month. Contact TRS-80 Users Group of Eastern Mass., c/o Miller, 61 Lake Shore Road, Natick, MA 01760; (617) 653-6136.



JUNE 6-8 ANAHEIM, CA

The 1978 National Computer Conference will feature a Personal Computing Festival to take place June 6-8 at the Disneyland Hotel complex in Anaheim, CA. Both one-day and three-day registrations will be available for the Festival. Information on NCC 78 may be obtained from AFIPS Headquarters, 210 Summit Ave, Montvale, NJ 07645 or by calling (201) 391-9810.

JUNE 23-25 DETROIT, MI

The MACC Computerfest '78 will be held at the Detroit Plaza Hotel June 23-25, 1978. Conference Chairperson is Jim Rarus. Write to PO Box 9578 North End Station, Detroit, MI; (313) 775-5320.

JULY 22-23 ARLINGTON, VA

Several thousand people are expected to attend Amateur Computing 78, a July 22-23 microcomputer festival to be held at the Sheraton National Motor Hotel in Arlington, VA. This event is being sponsored by AMRAD, a technically oriented club of radio amateurs and computerists in the Washington, DC Area. For further information, write AMRAD, Box 682, McLean, VA 22101.

AUG 22-25 BELLAIRE, MI

The International Conference on Parallel Processing, sponsored by IEEE Computer Society and Wayne State University, will be held August 22-25 in Bellaire, Michigan. Contact Professor G.J. Lipouski, Deptartment of Electrical Engineering, University of Texas, Austin, TX 78712.

OCT 10-12 SAN FRANCISCO, CA

The third USA-Japan Computer Conference will be held October 10-12, 1978 in San Francisco. This marks the first time this gathering is to be held on American soil. Contact Professor Edward J. McCluskey, Digital System Laboratory, Stanford University, Stanford, CA 94305.